Surviving congestion in geo-distributed storage systems

Brian Cho *University of Illinois at Urbana-Champaign*
Marcos K. Aguilera *Microsoft Research Silicon Valley*

Geo-distributed data centers

- Web applications increasingly deployed across geo-distributed data centers
  - e.g., social networks, online stores, messaging
- App data replicated across data centers
  - Disaster tolerance
  - Access locality

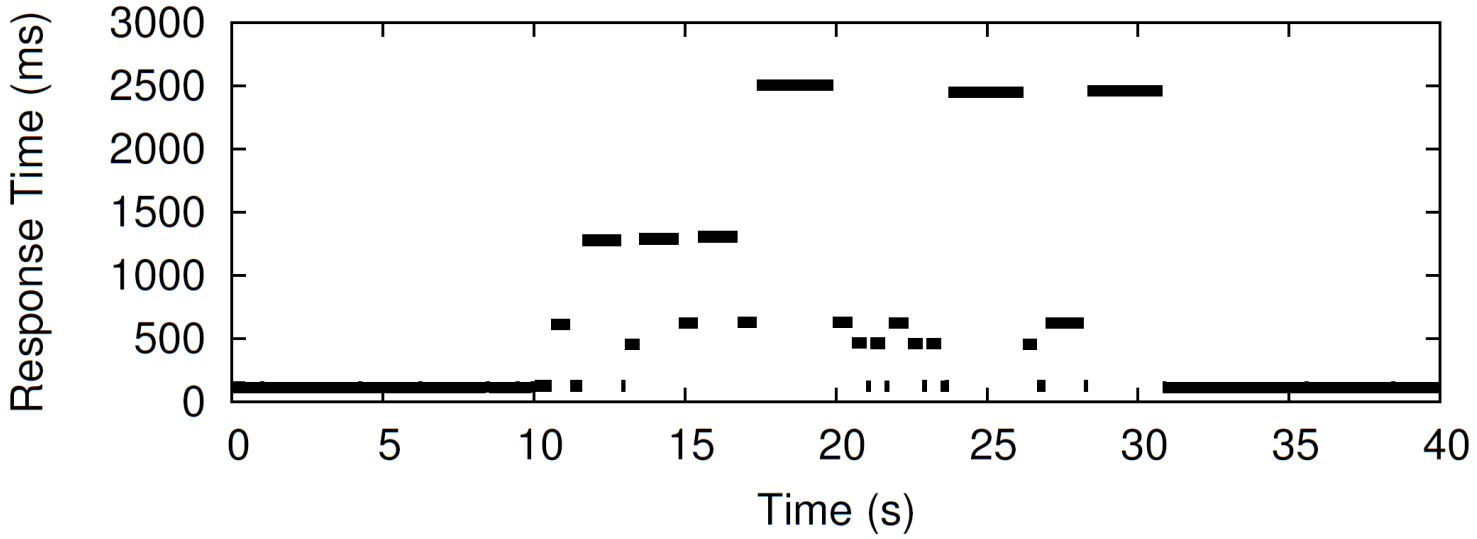
Congestion between geo-distributed data centers

- Limited bandwidth between data centers
  - e.g., leased lines, MPLS VPN
  - Bandwidth is expensive: ~1K $/Mbps [SprintMPLS]
  - Provision for typical (not peak) usage
- Many machines in each data center

Congestion → Delay between geo-distributed data centers

- Congestion can cause significant delays
  - TCP messaging increases to order-of-seconds (Figure)
  - Observed across Amazon EC2 data centers [Kraska et al]
- Users do not tolerate delays (<1s) [Nielsen]

FIGURE: RPC round trip delay under congestion (10-30s)



Replication techniques applied to geo-distributed data centers

- Weak consistency
  - e.g., Amazon Dynamo, Yahoo PNUTS, COPS
  - Good performance: updates can be propagated asynchronously
  - Semantics undesirable in some cases (e.g., writes get re-ordered across replicas)

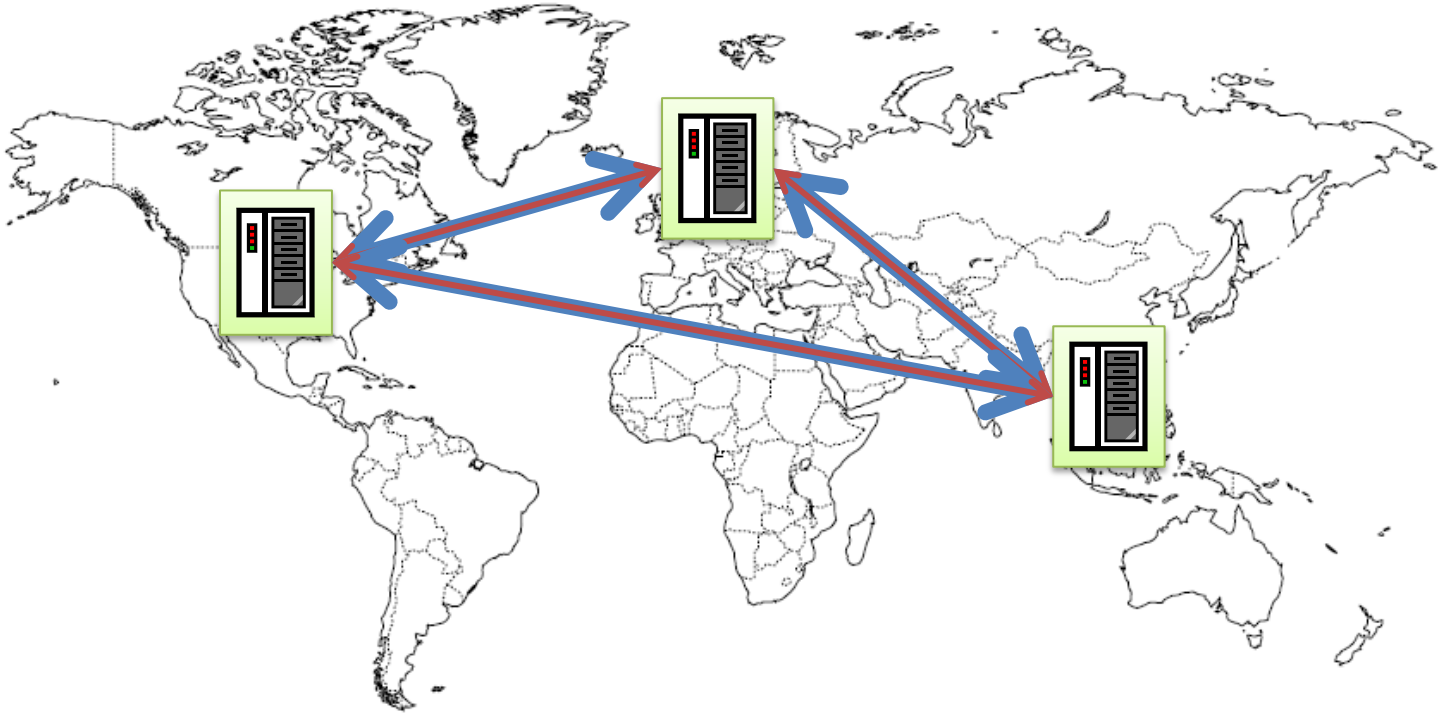
- Strong consistency
  - e.g., ABD, Paxos, available in Google Megastore, Amazon SimpleDB
  - Avoids the many problems of weak consistency
  - Must wait for updates to propagate across data centers
  - App delay requirements difficult to meet under congestion

Contributions

- Vivace: a strongly consistent key-value store that is resilient to congestion across geo-distributed data centers
- Approach
  - New algorithms send small amount of critical information across data centers in separate prioritized messages
- Challenges
  - Still provide strong consistency
  - Keep prioritized messages small
  - Avoid delay overhead in absence of congestion

Vivace algorithms

- Enhance previous strongly consistent algorithms
- Prioritize small amount of critical information across sites



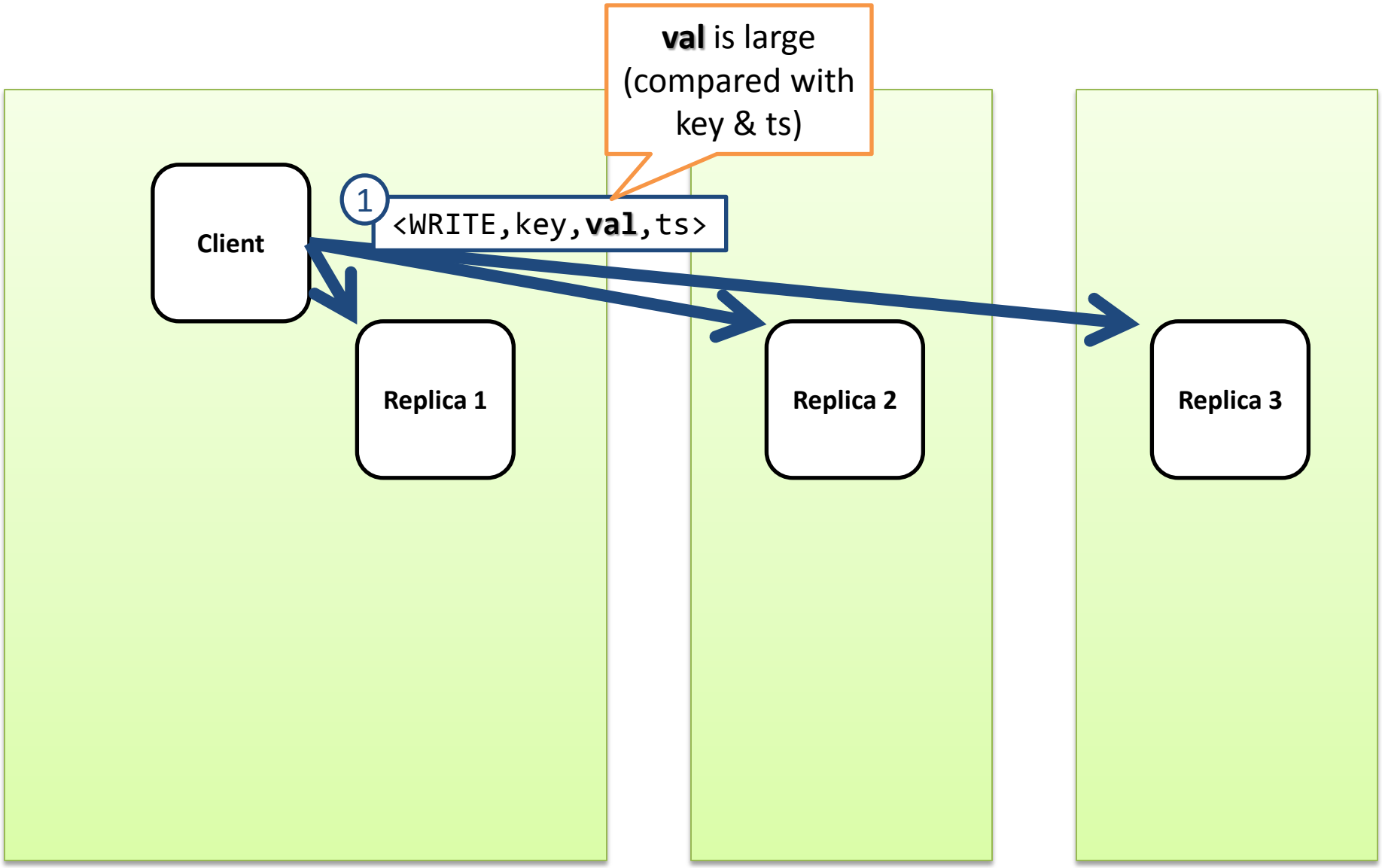
Vivace algorithms

- Enhance previous strongly consistent algorithms
- Prioritize small amount of critical information across sites

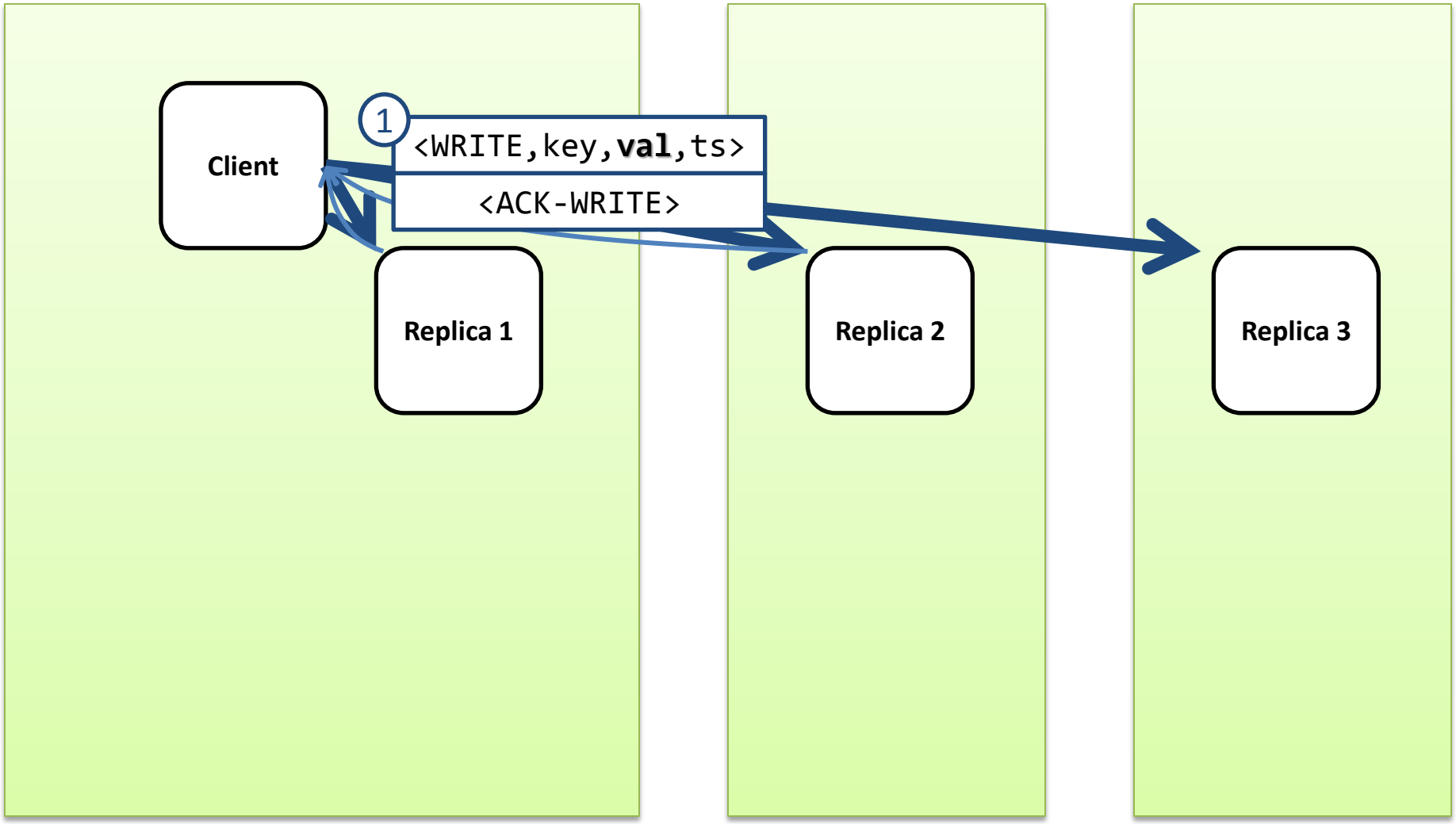
Two algorithms:

1. Read/write algorithm
   - Very simple
   - Based on traditional quorum algorithm [ABD]
   - Linearizable read() and write()
   - read() contains a write-back phase
2. State machine replication algorithm
   - More complex, details in paper

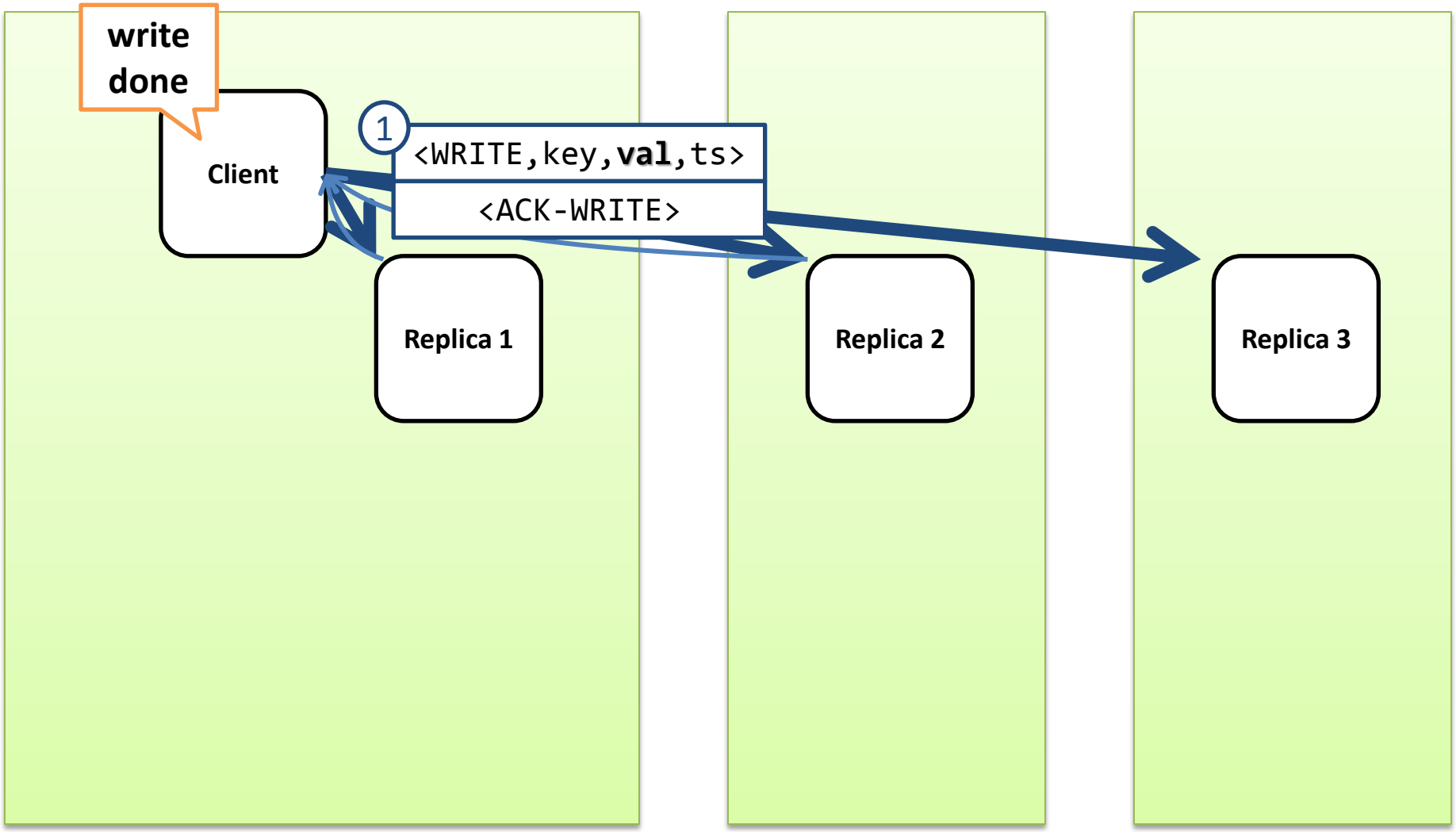
Traditional quorum algorithm: write



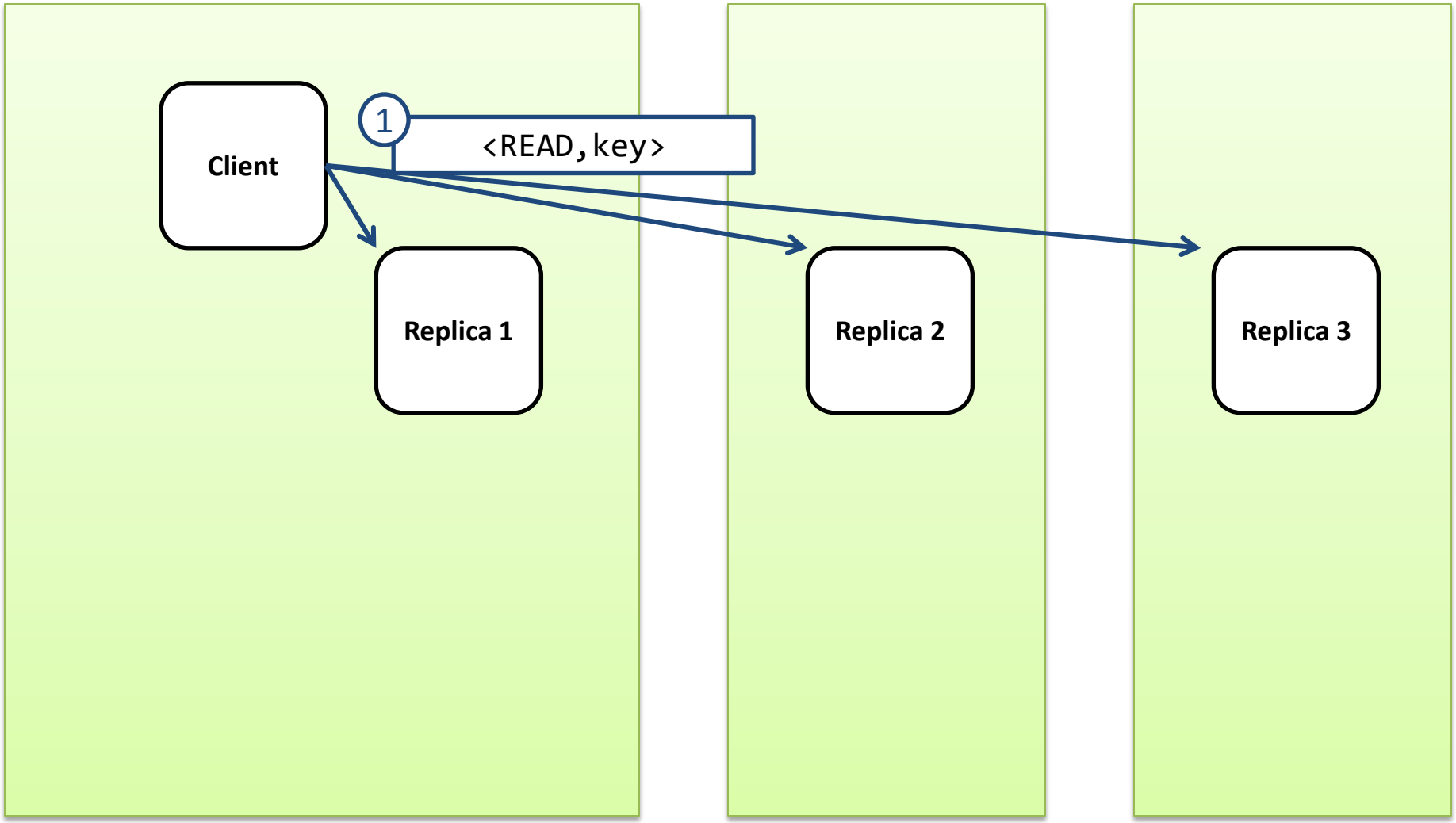
Traditional quorum algorithm: write



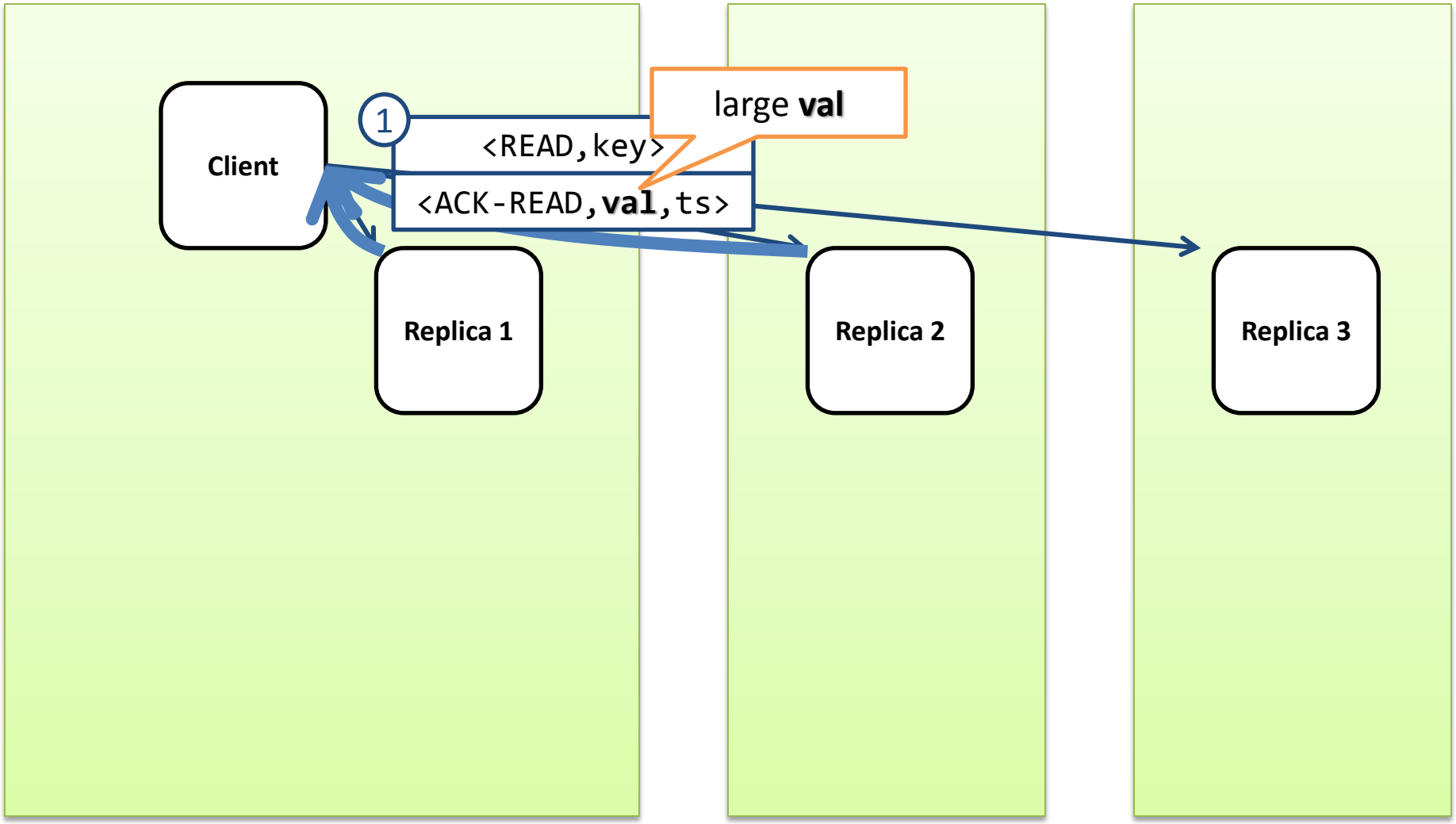
Traditional quorum algorithm: write



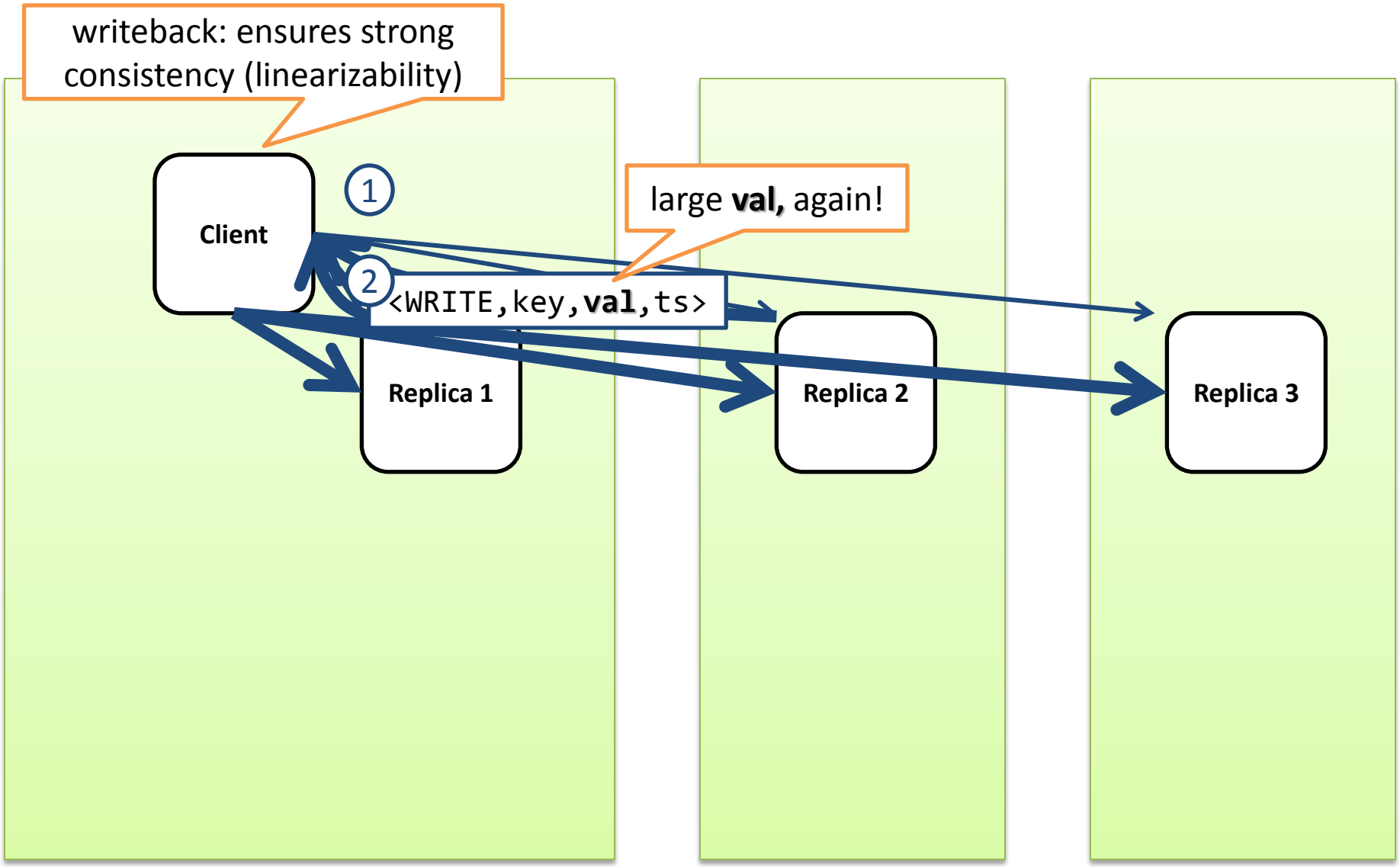
Traditional quorum algorithm: read



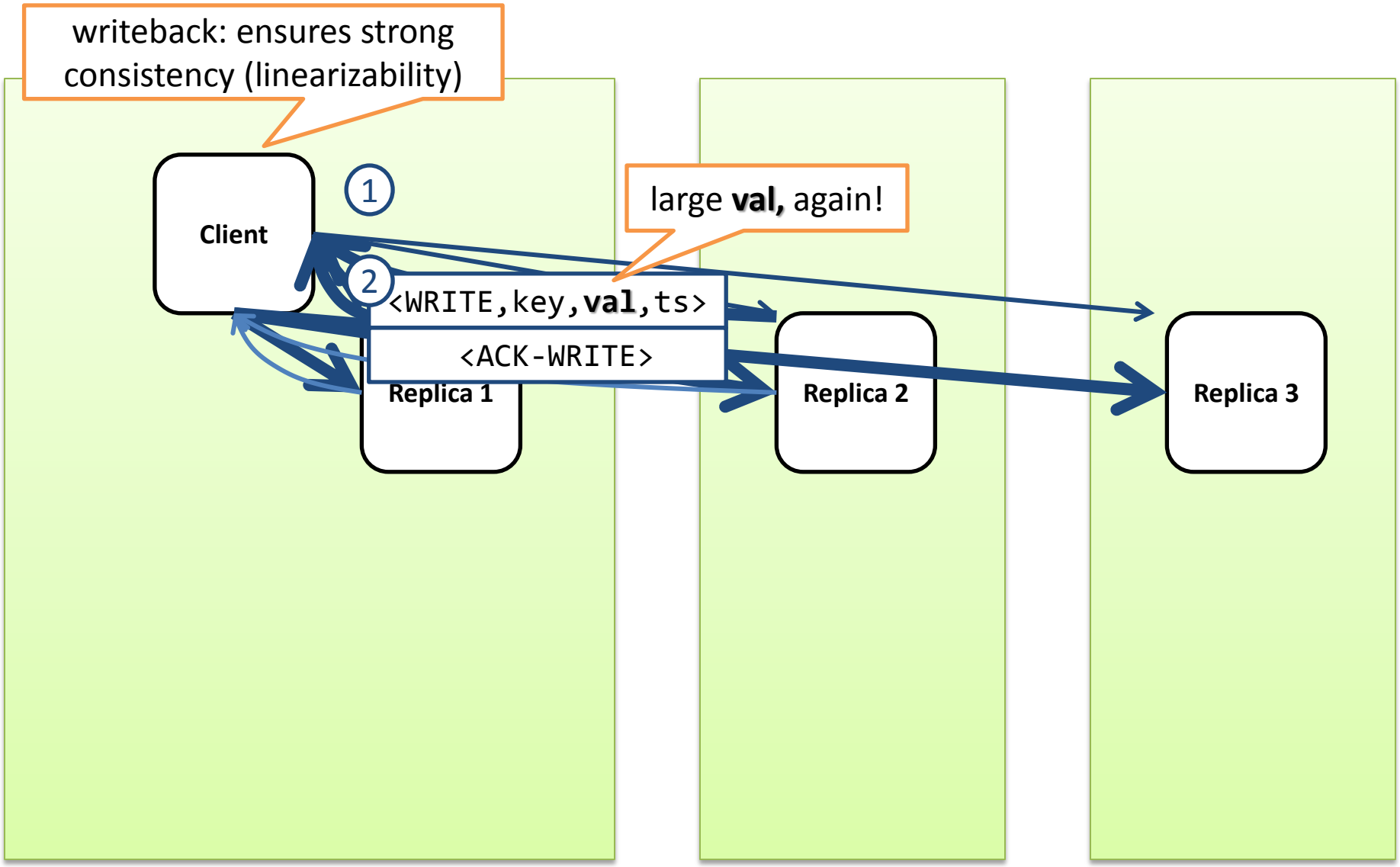
Traditional quorum algorithm: read



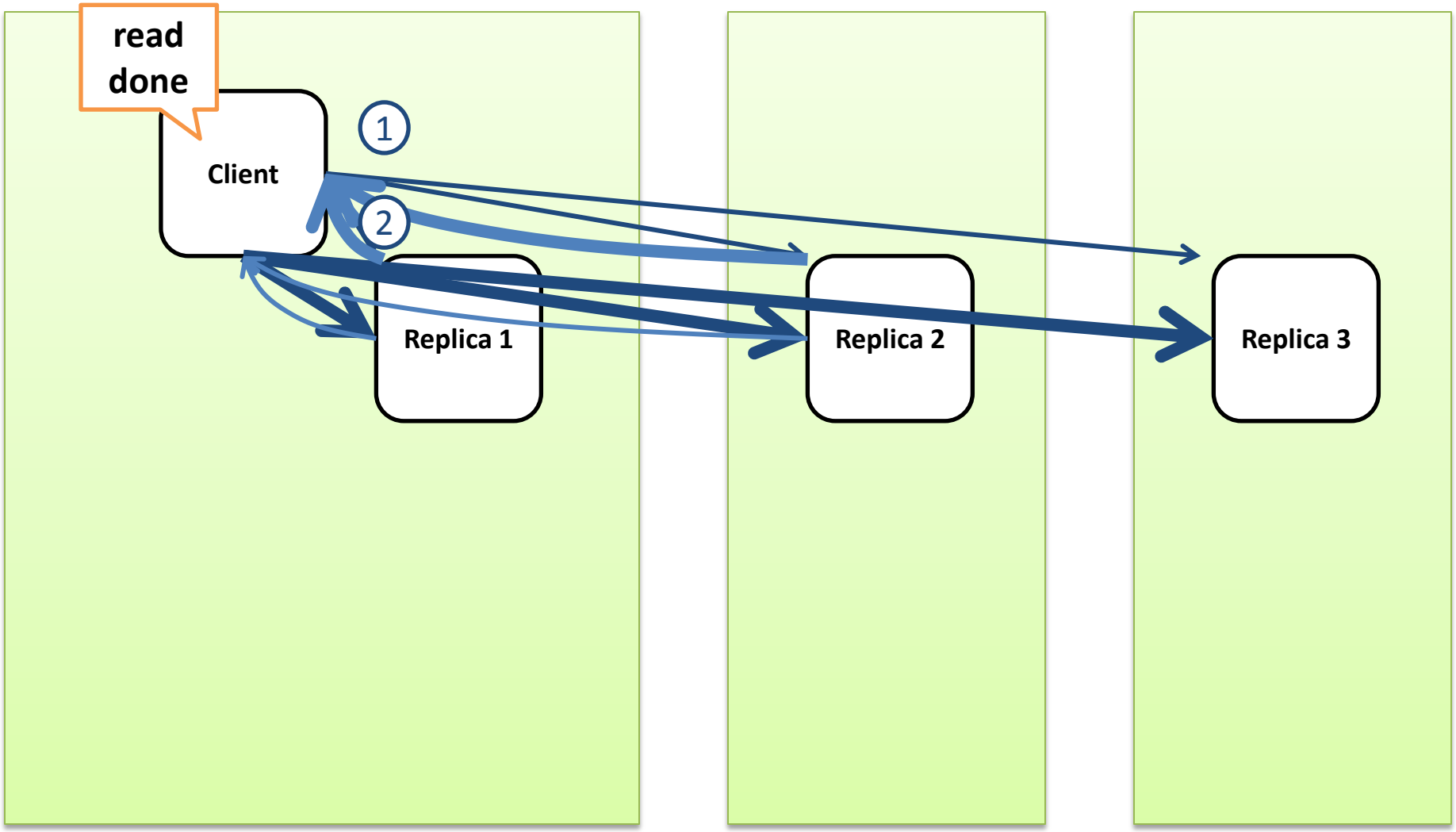
Traditional quorum algorithm: read



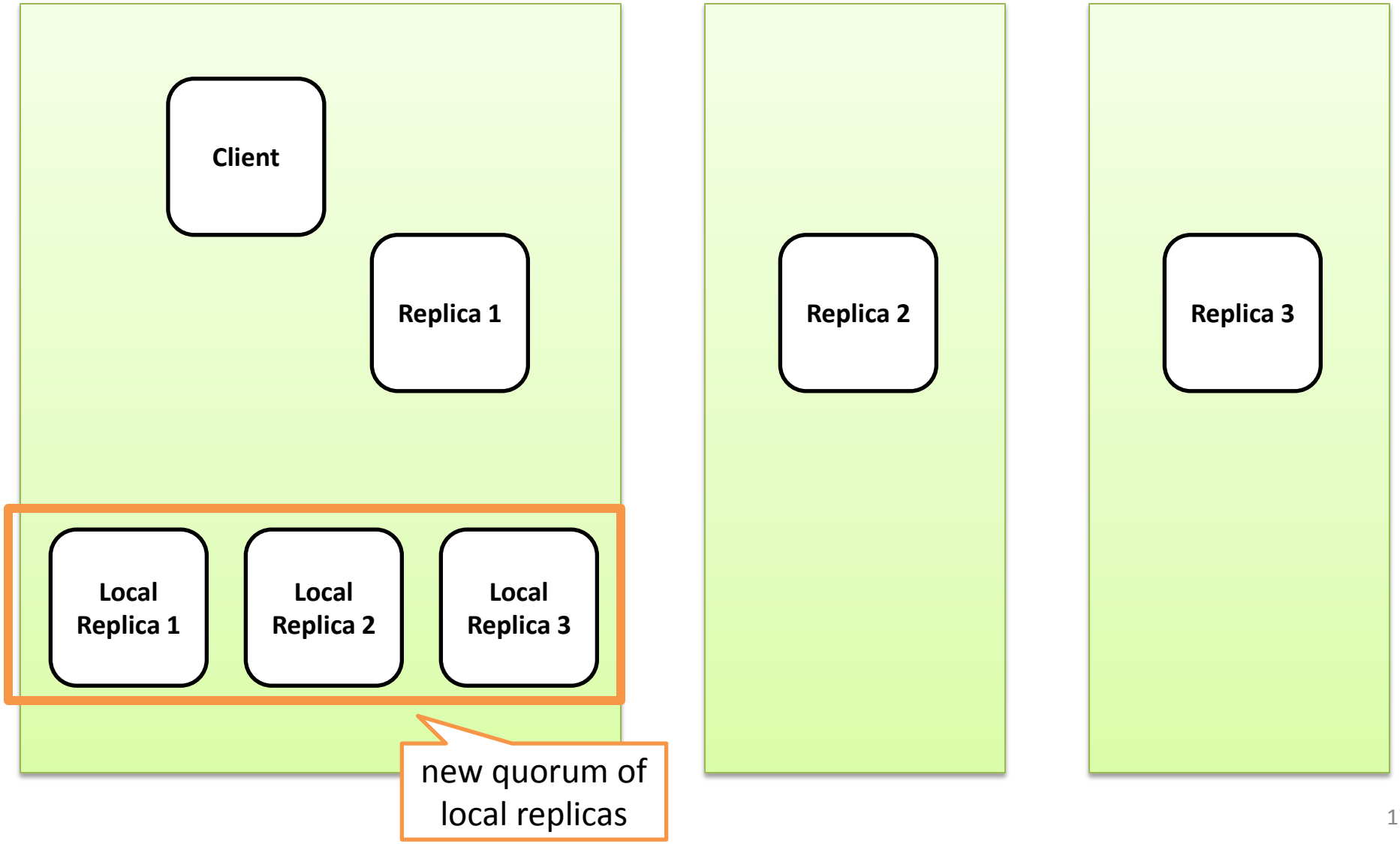
Traditional quorum algorithm: read



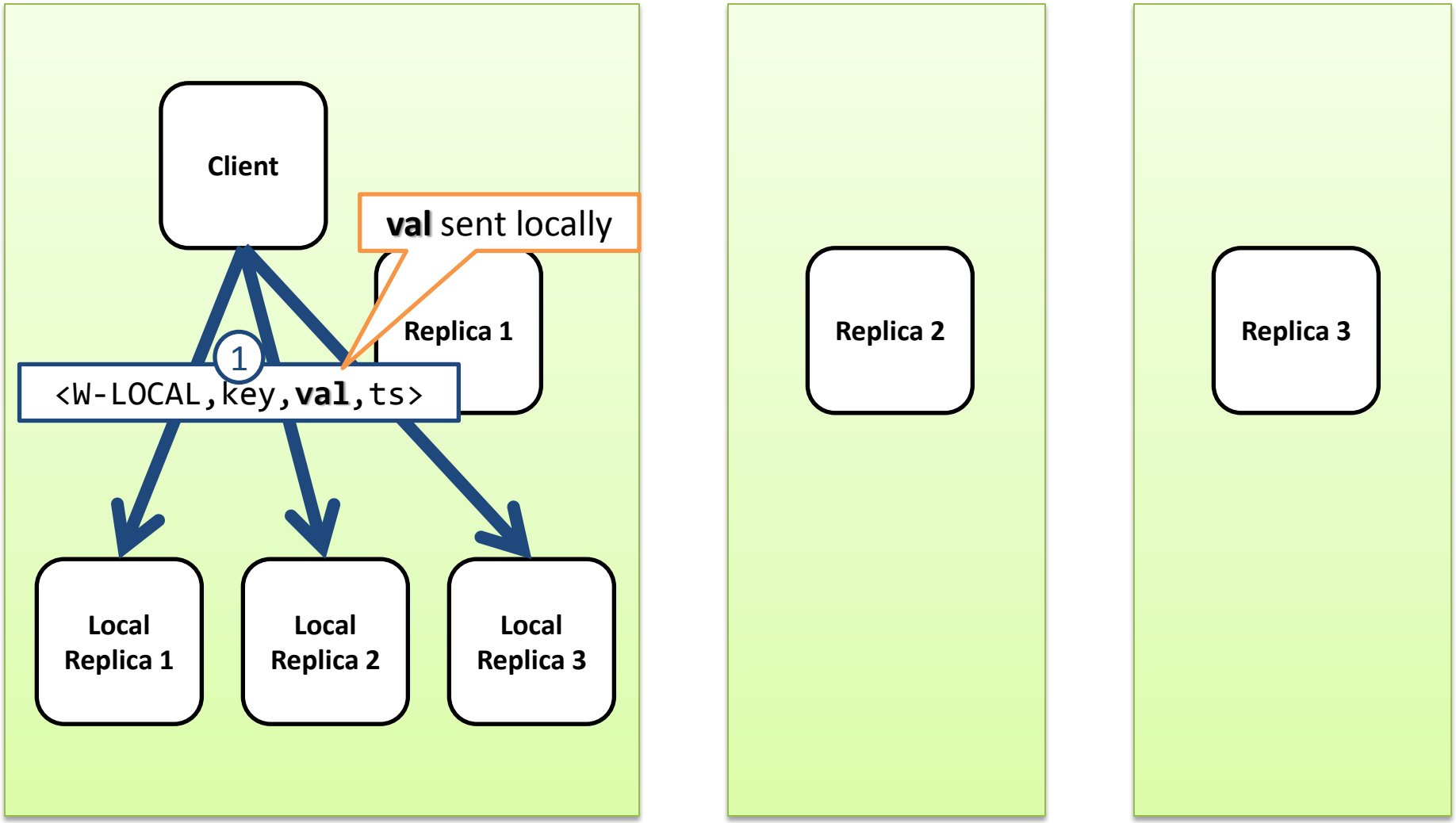
Traditional quorum algorithm: read



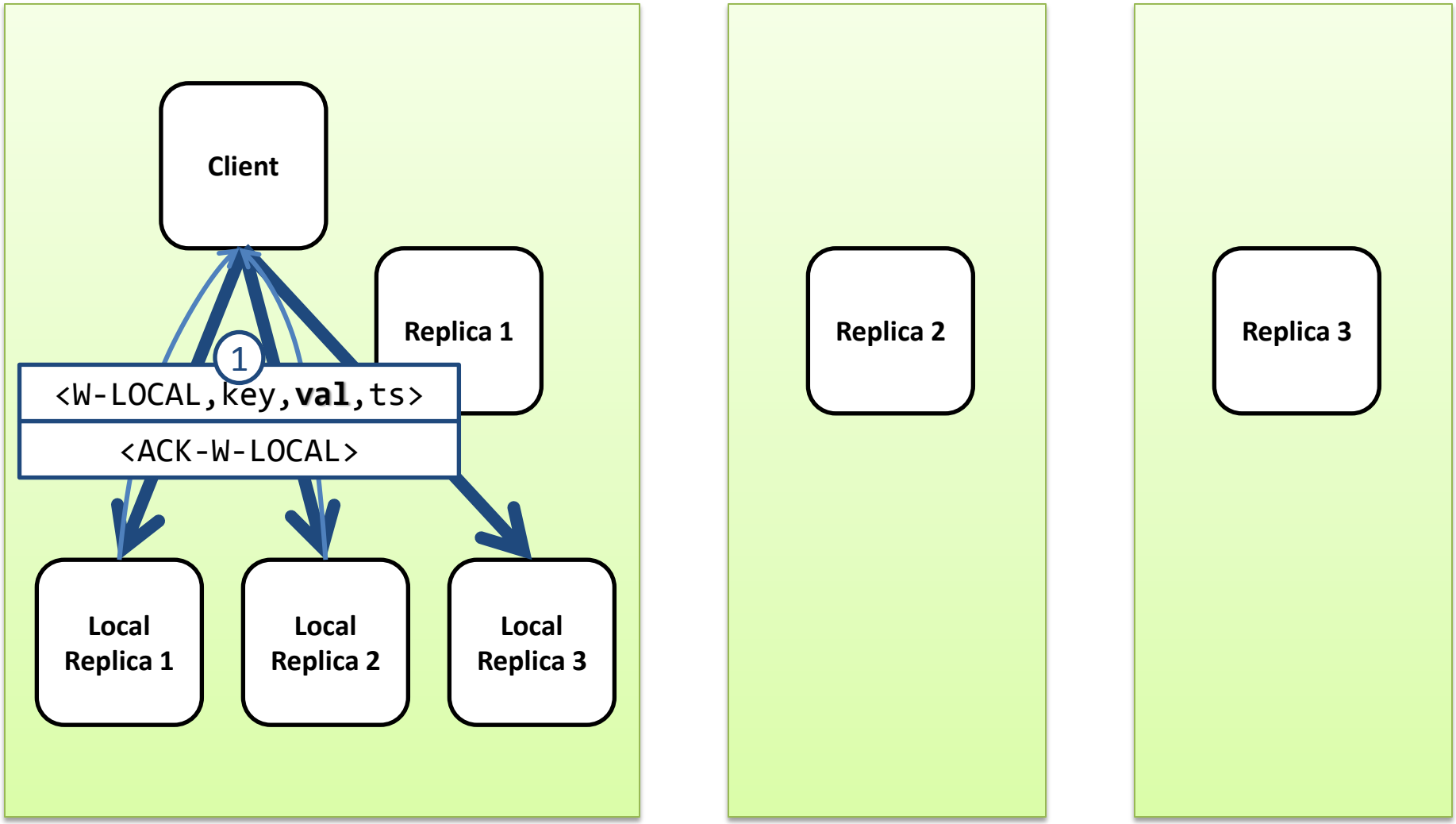
Vivace: write



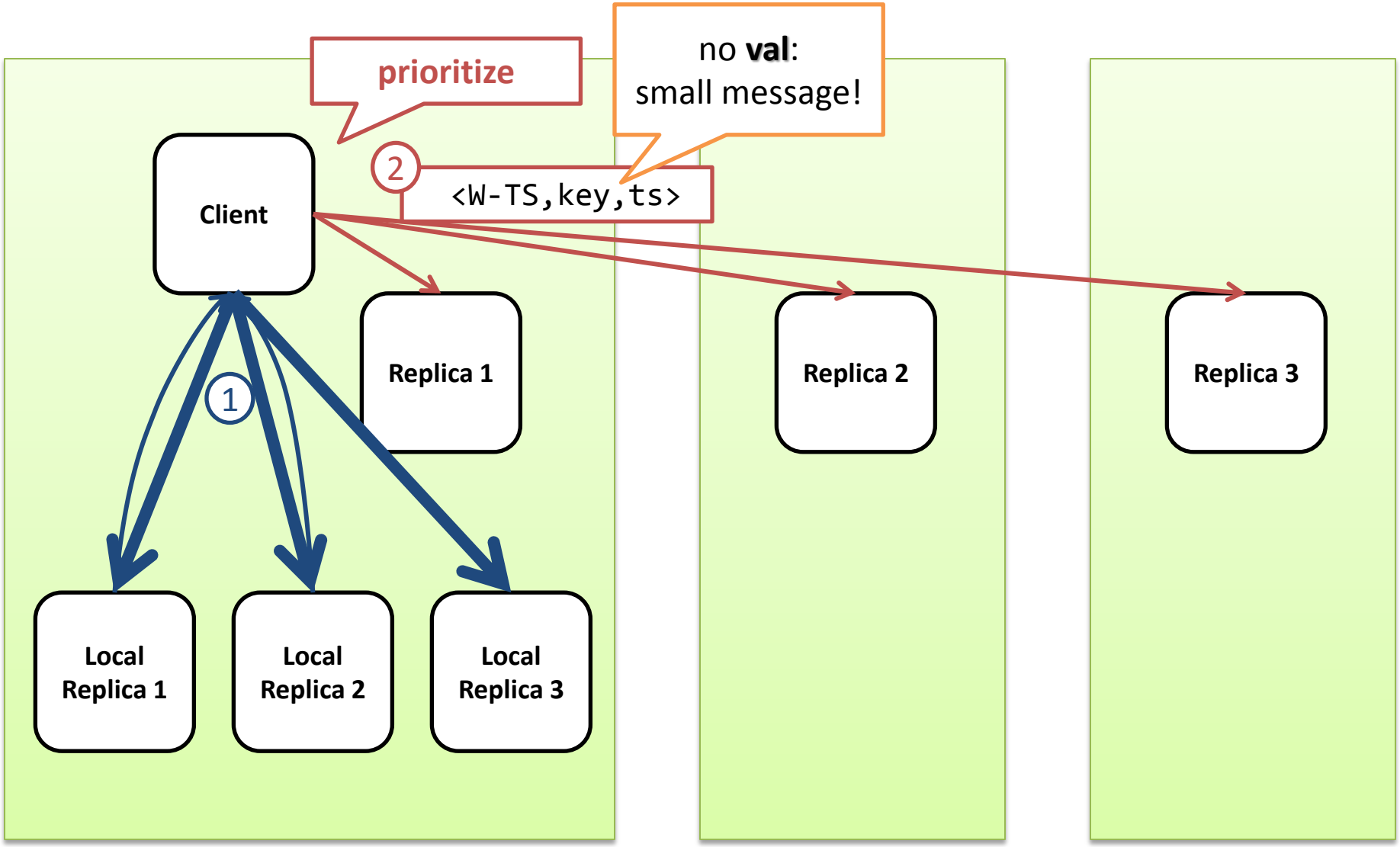
Vivace: write



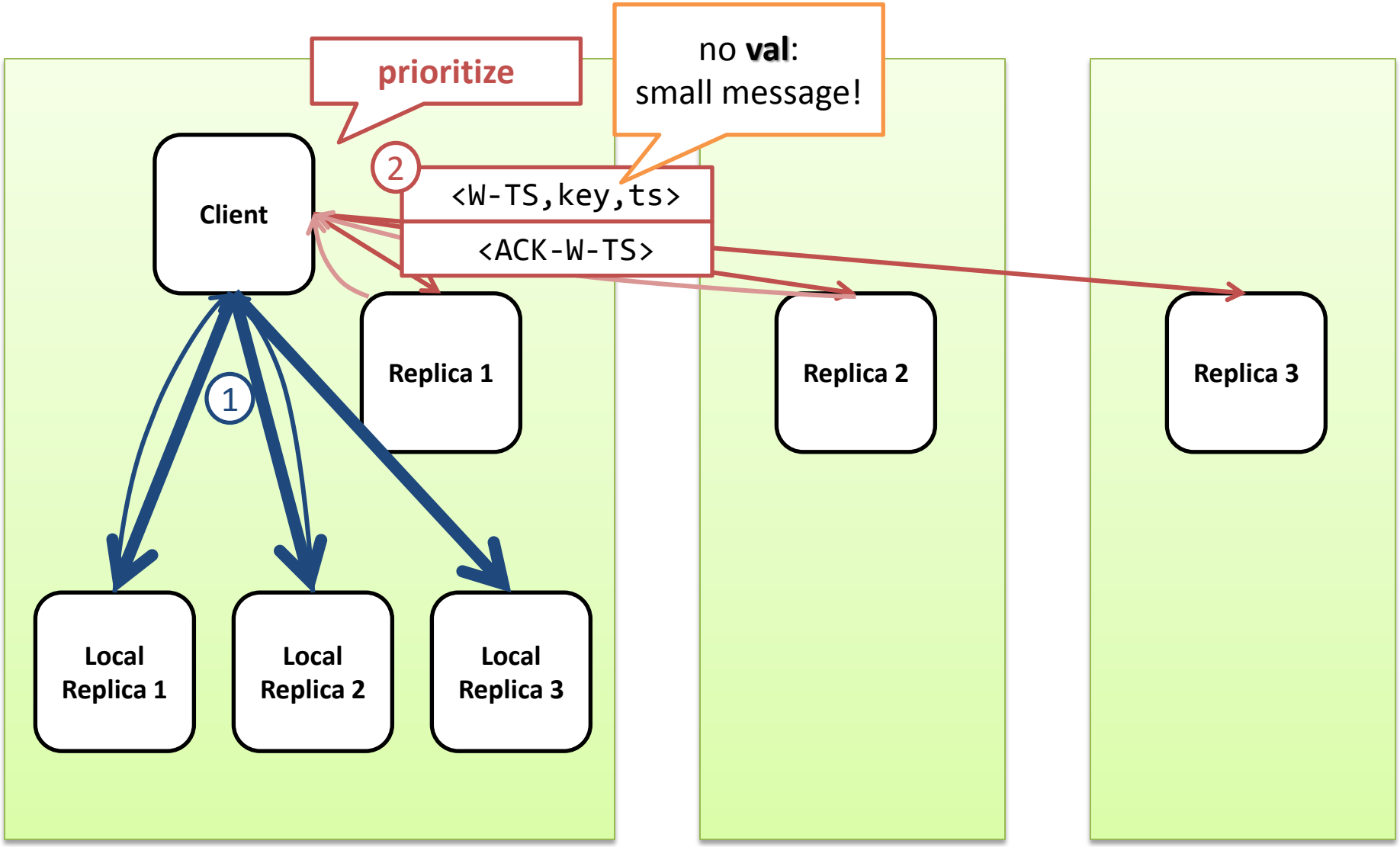
Vivace: write



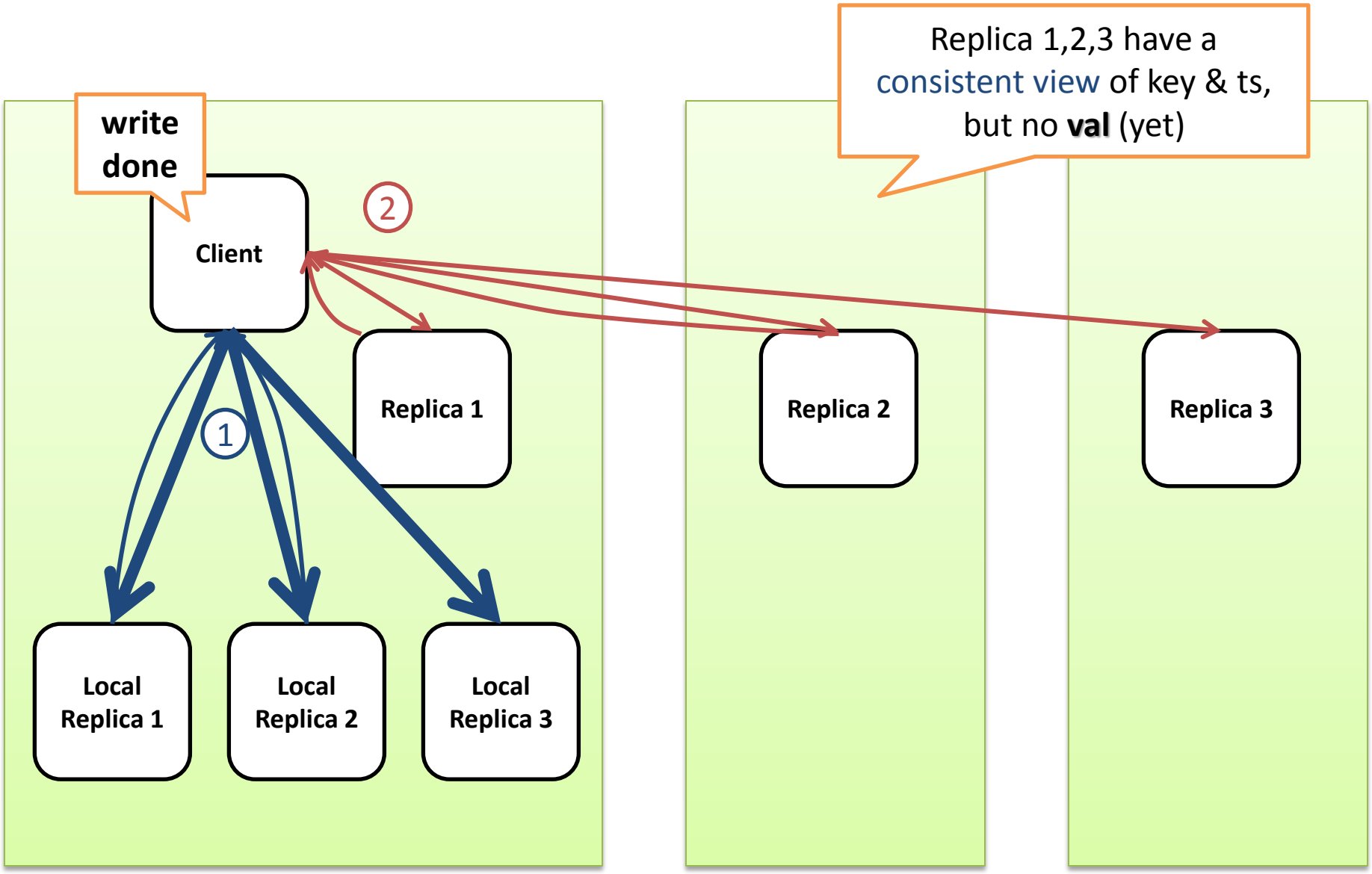
Vivace: write



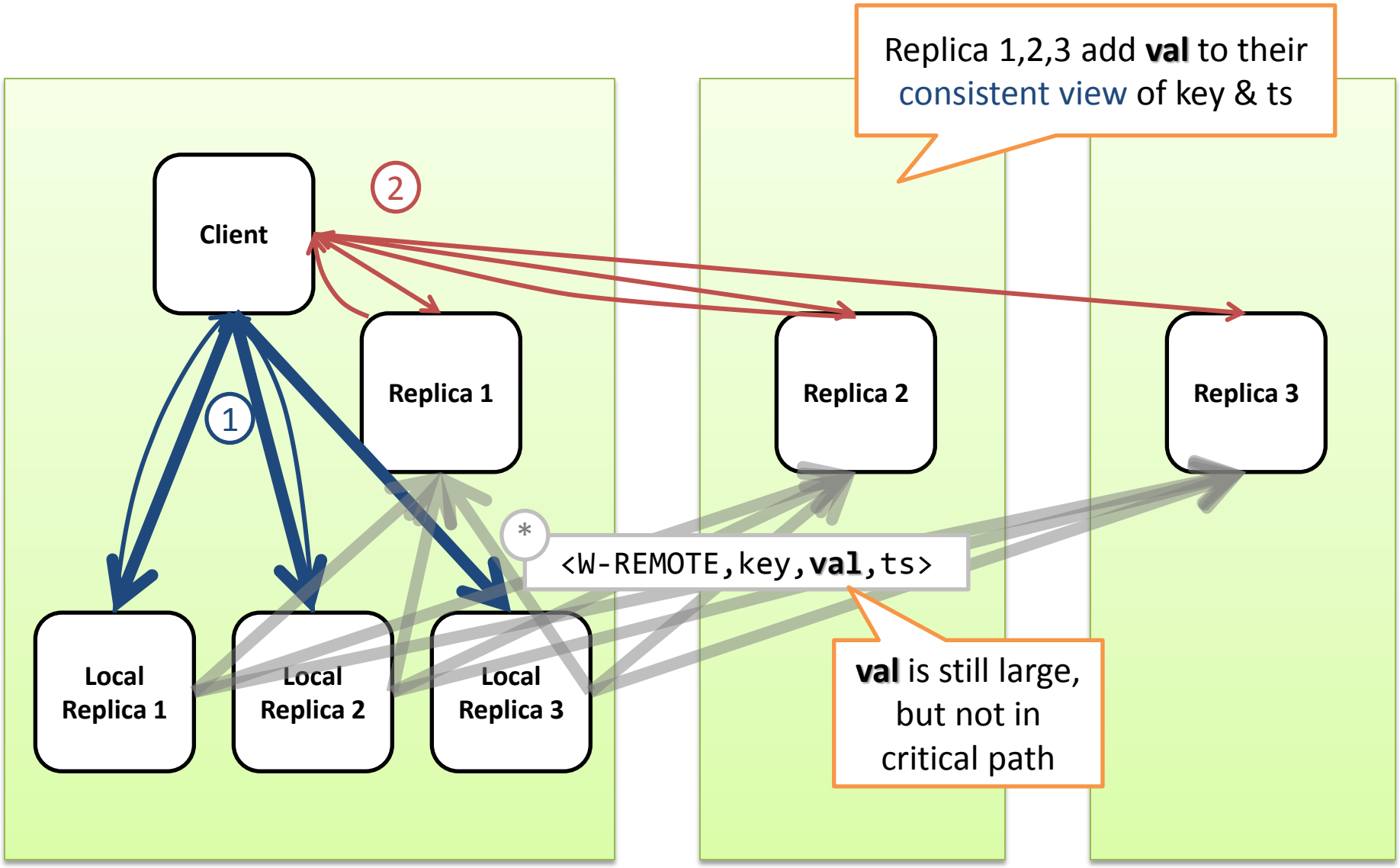
Vivace: write



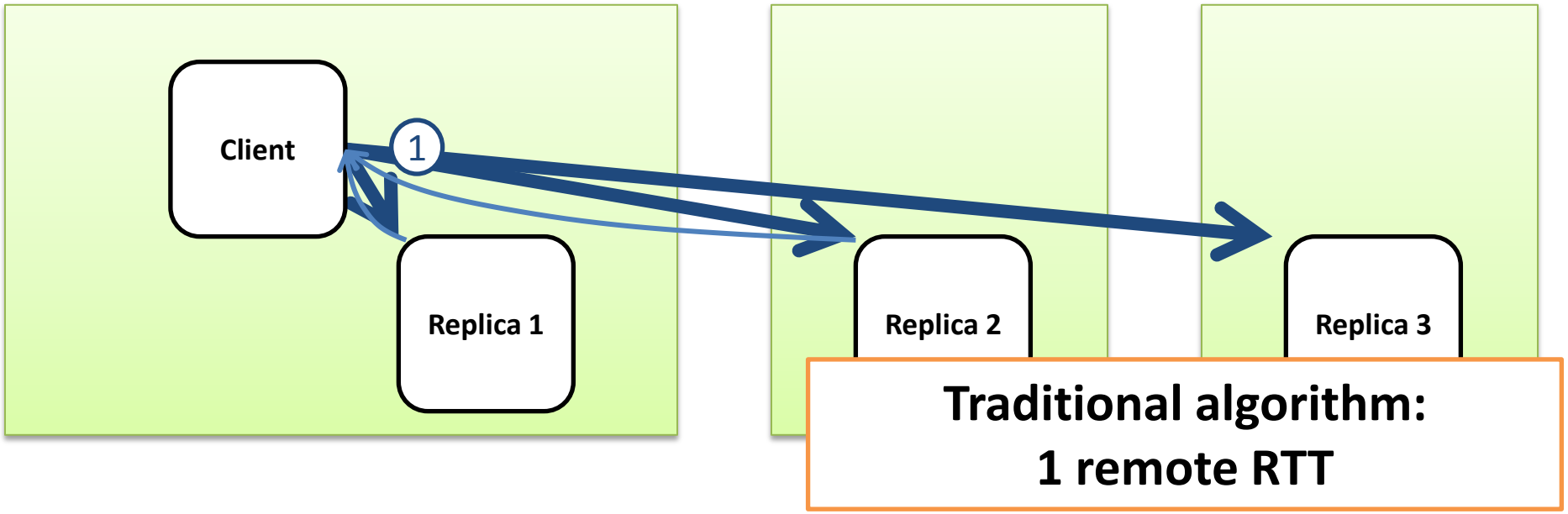
Vivace: write

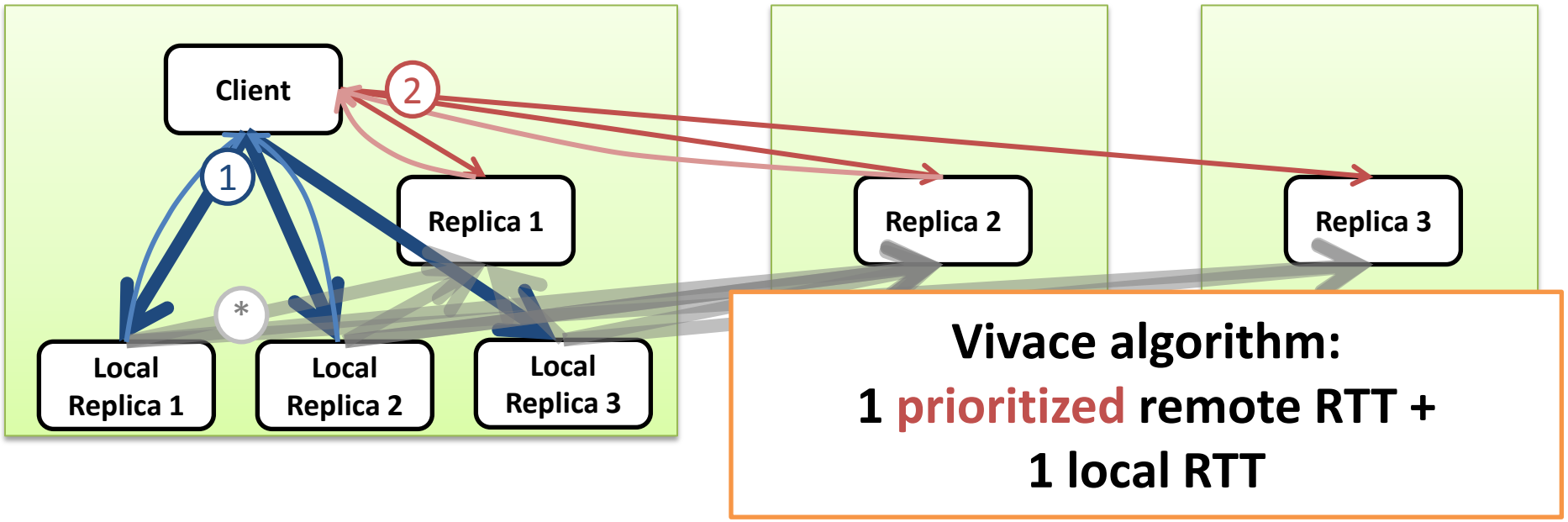


Vivace: write

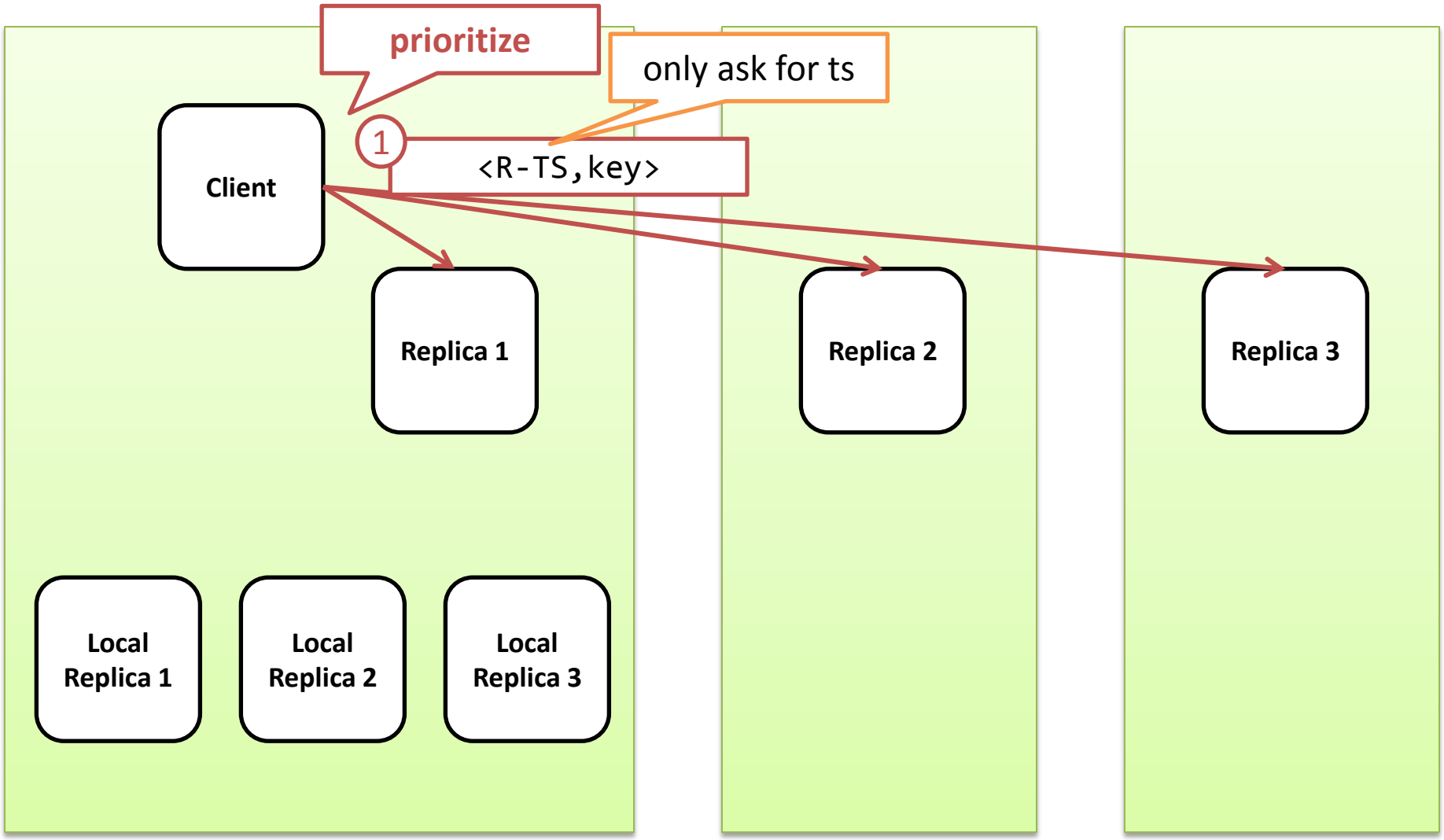


write comparison

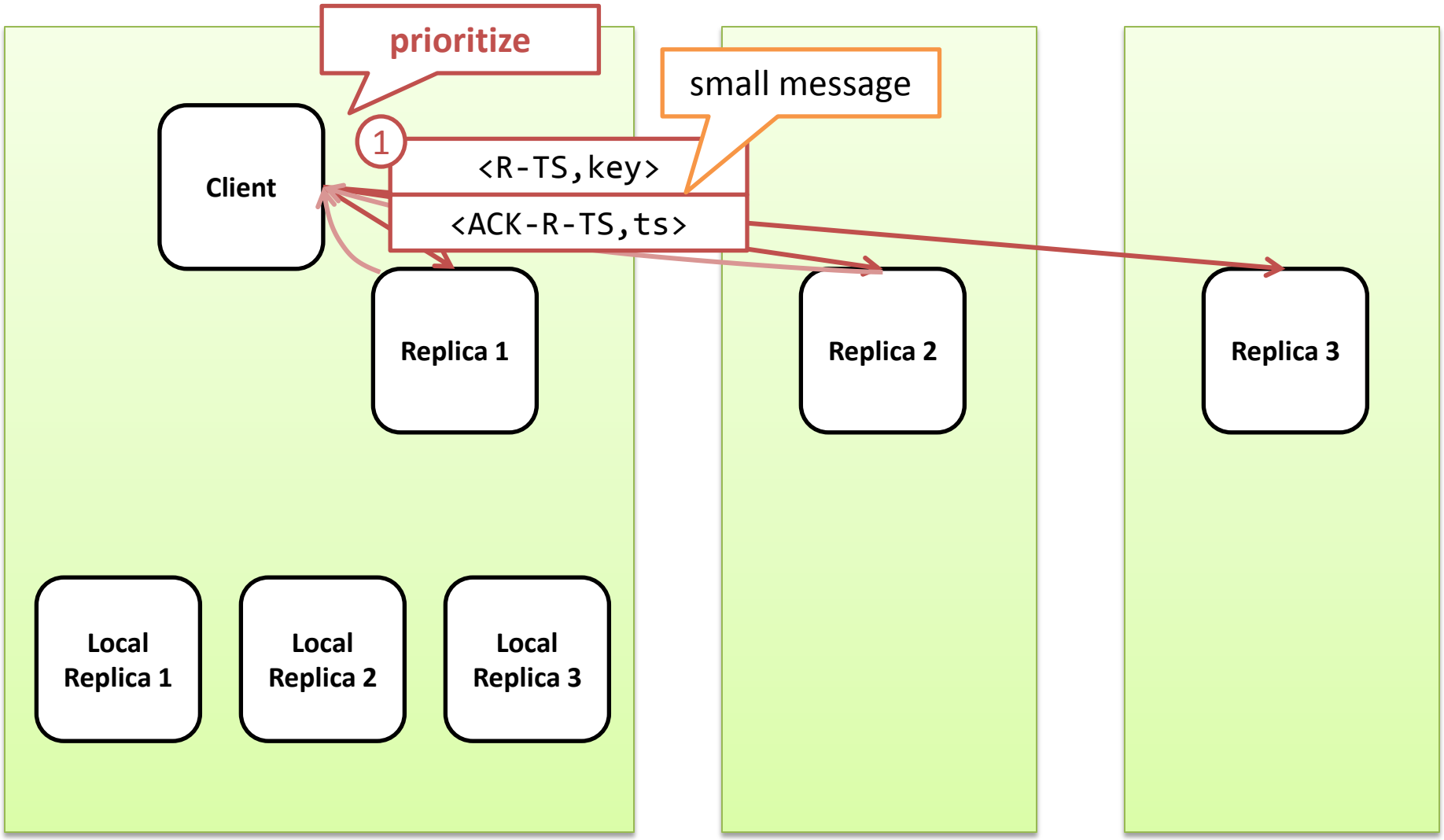




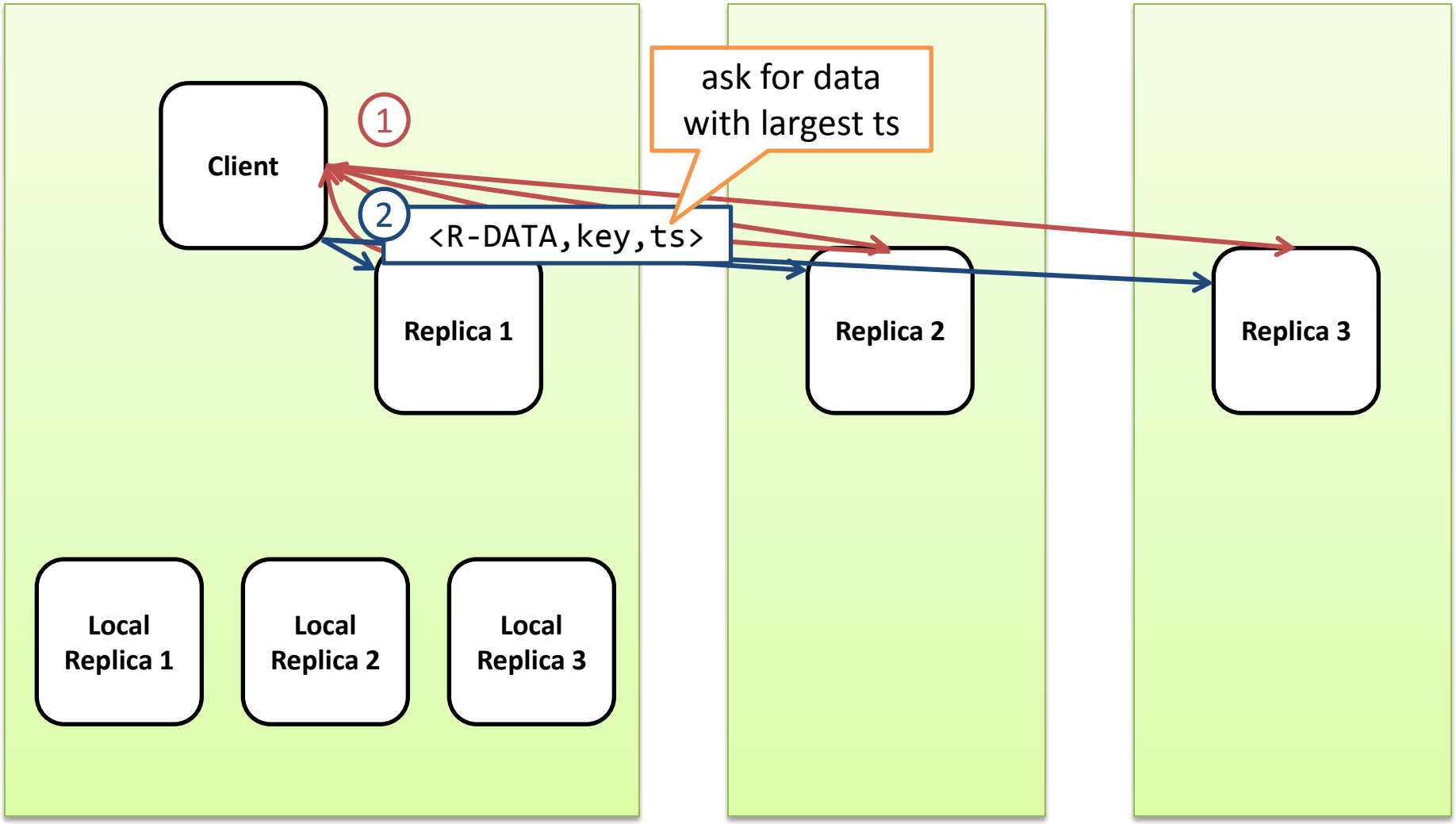
Vivace: read


prioritize
only ask for ts
1
<R-TS,key>
Client
Replica 1
Replica 2
Replica 3
Local Replica 1
Local Replica 2
Local Replica 3

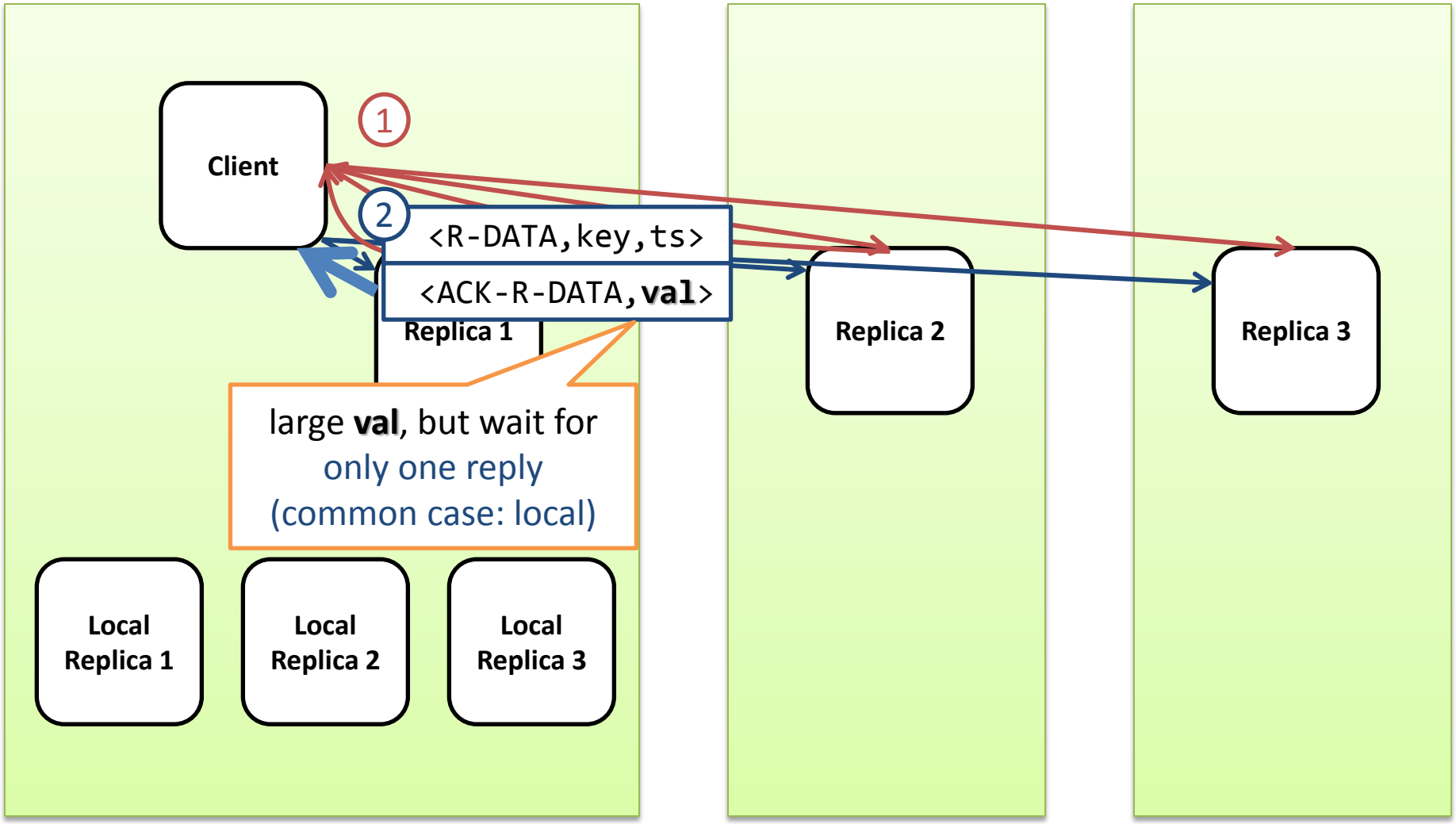
Vivace: read


prioritize
small message
1
Client
<R-TS,key>
<ACK-R-TS,ts>
Replica 1
Replica 2
Replica 3
Local Replica 1
Local Replica 2
Local Replica 3

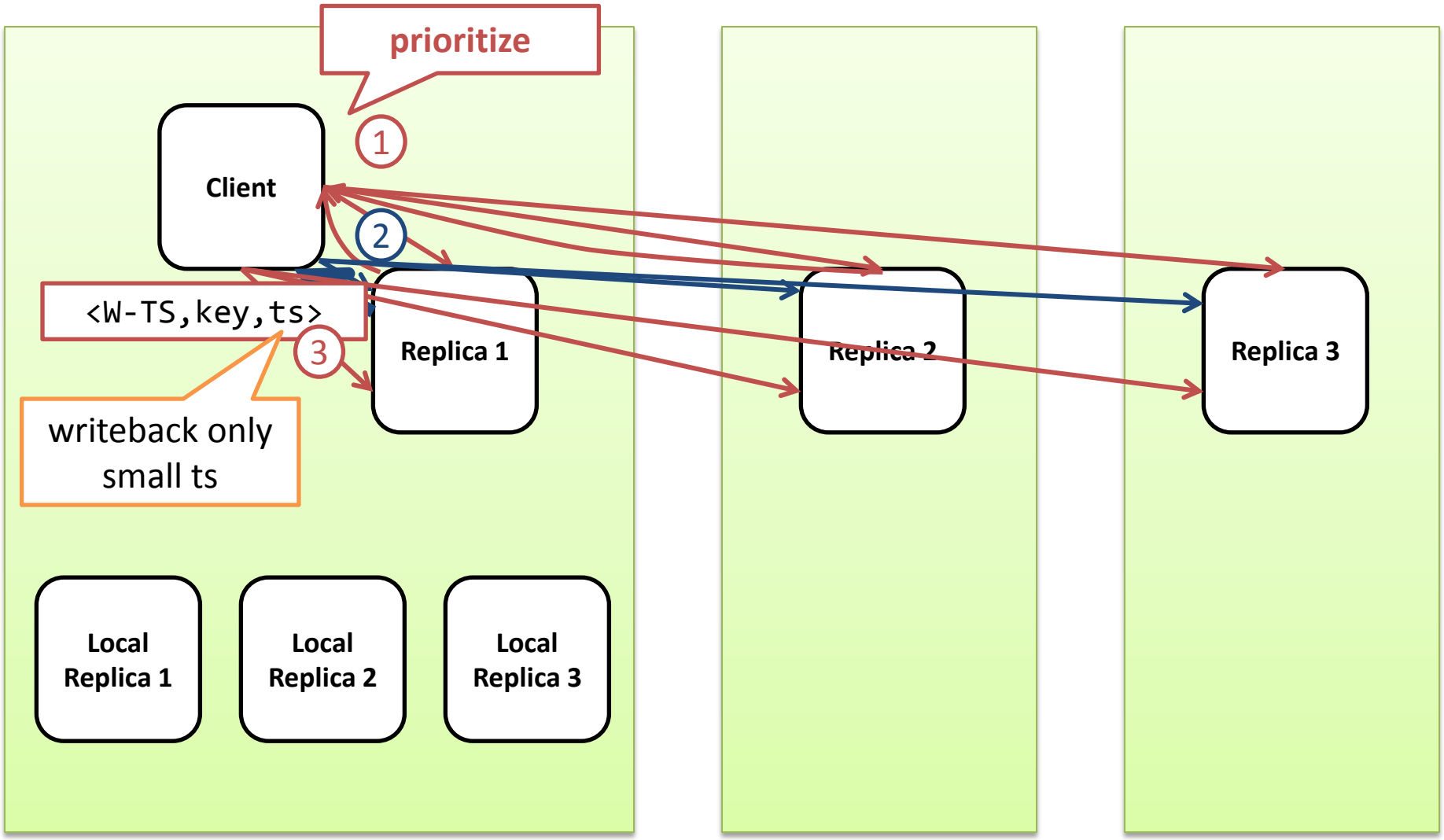
Vivace: read


Client
1
2
<R-DATA,key,ts>
ask for data
with largest ts
Replica 1
Replica 2
Replica 3
Local
Replica 1
Local
Replica 2
Local
Replica 3

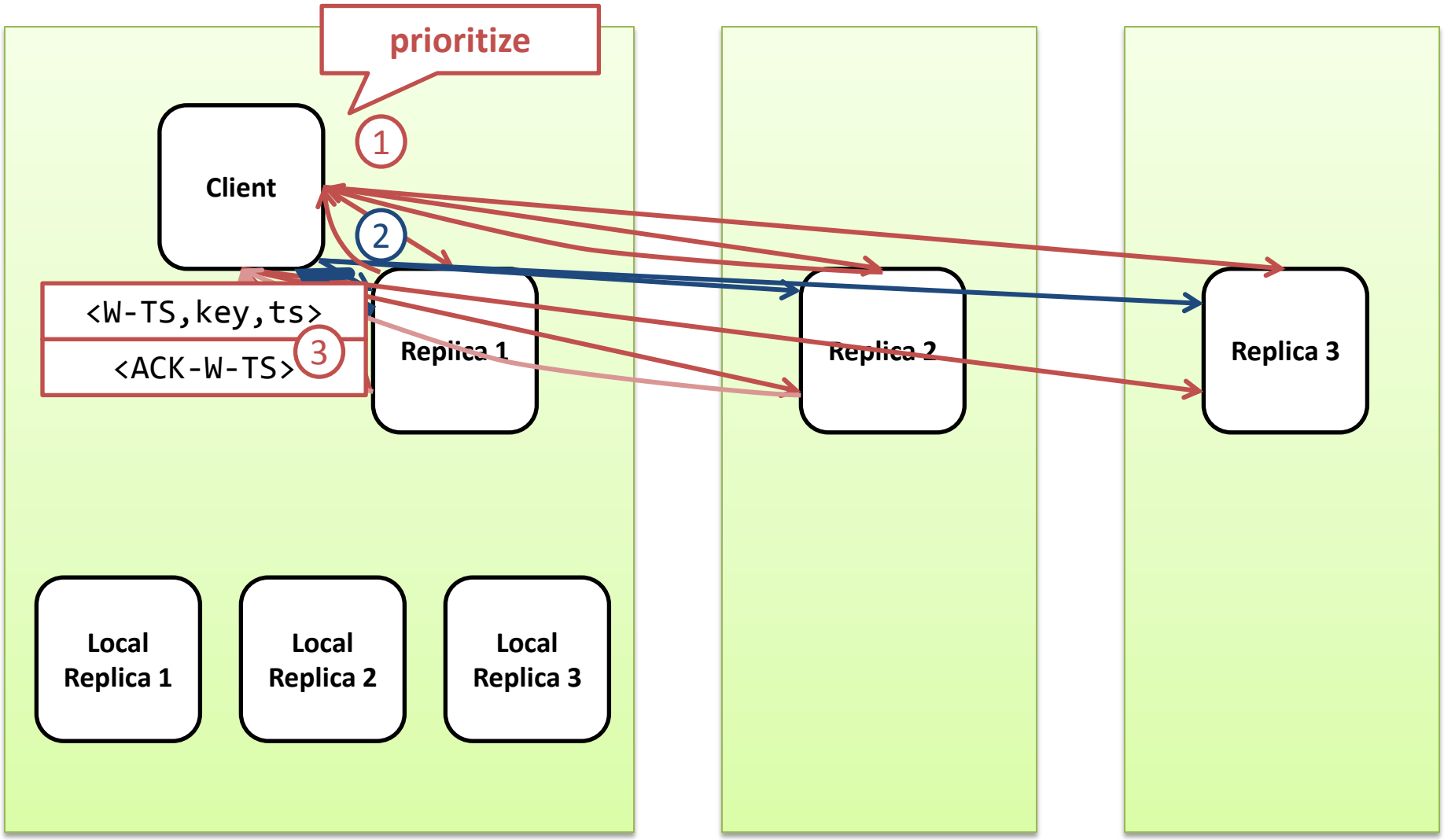
Vivace: read


Client
1
2
<R-DATA,key,ts>
<ACK-R-DATA,val>
Replica 1
Replica 2
Replica 3
large val, but wait for
only one reply
(common case: local)
Local
Replica 1
Local
Replica 2
Local
Replica 3

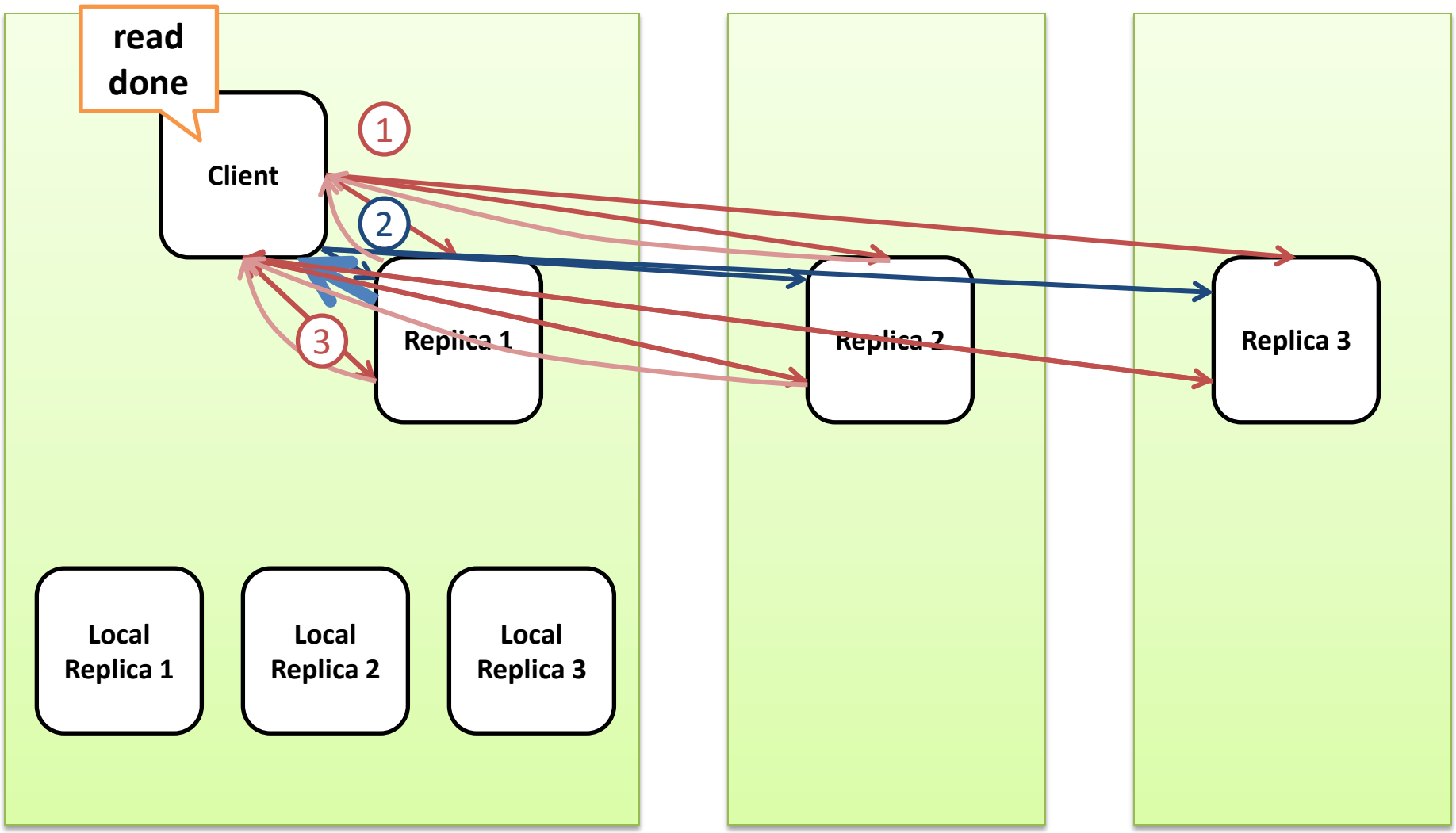
Vivace: read


prioritize
1
Client
2
<W-TS,key,ts>
3
writeback only
small ts
Replica 1
Replica 2
Replica 3
Local
Replica 1
Local
Replica 2
Local
Replica 3

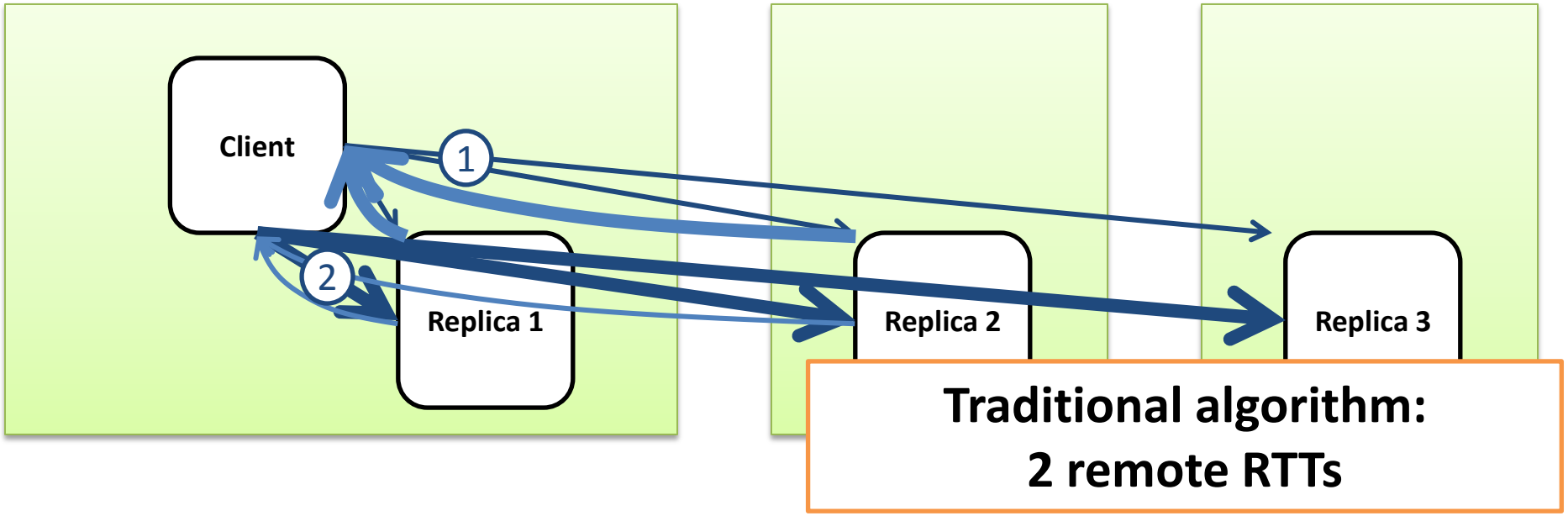
Vivace: read

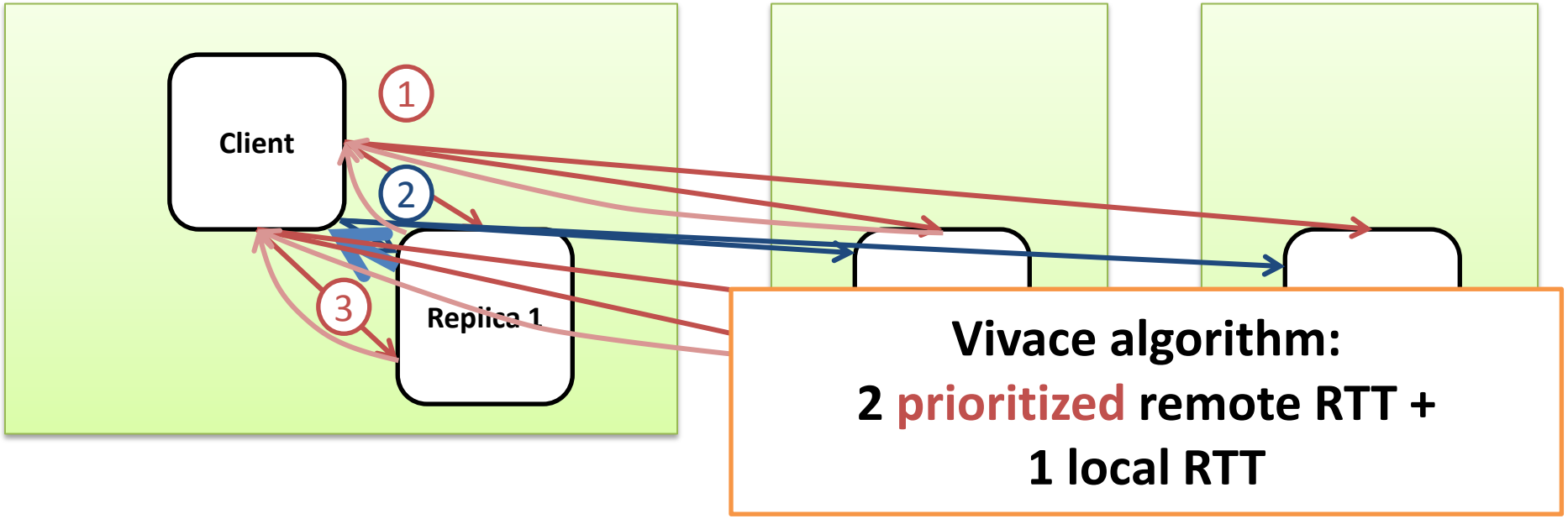

prioritize
1
Client
2
<W-TS,key,ts>
<ACK-W-TS>
3
Replica 1
Replica 2
Replica 3
Local Replica 1
Local Replica 2
Local Replica 3

Vivace: read


read done
Client
1
2
3
Replica 1
Replica 2
Replica 3
Local Replica 1
Local Replica 2
Local Replica 3

read comparison



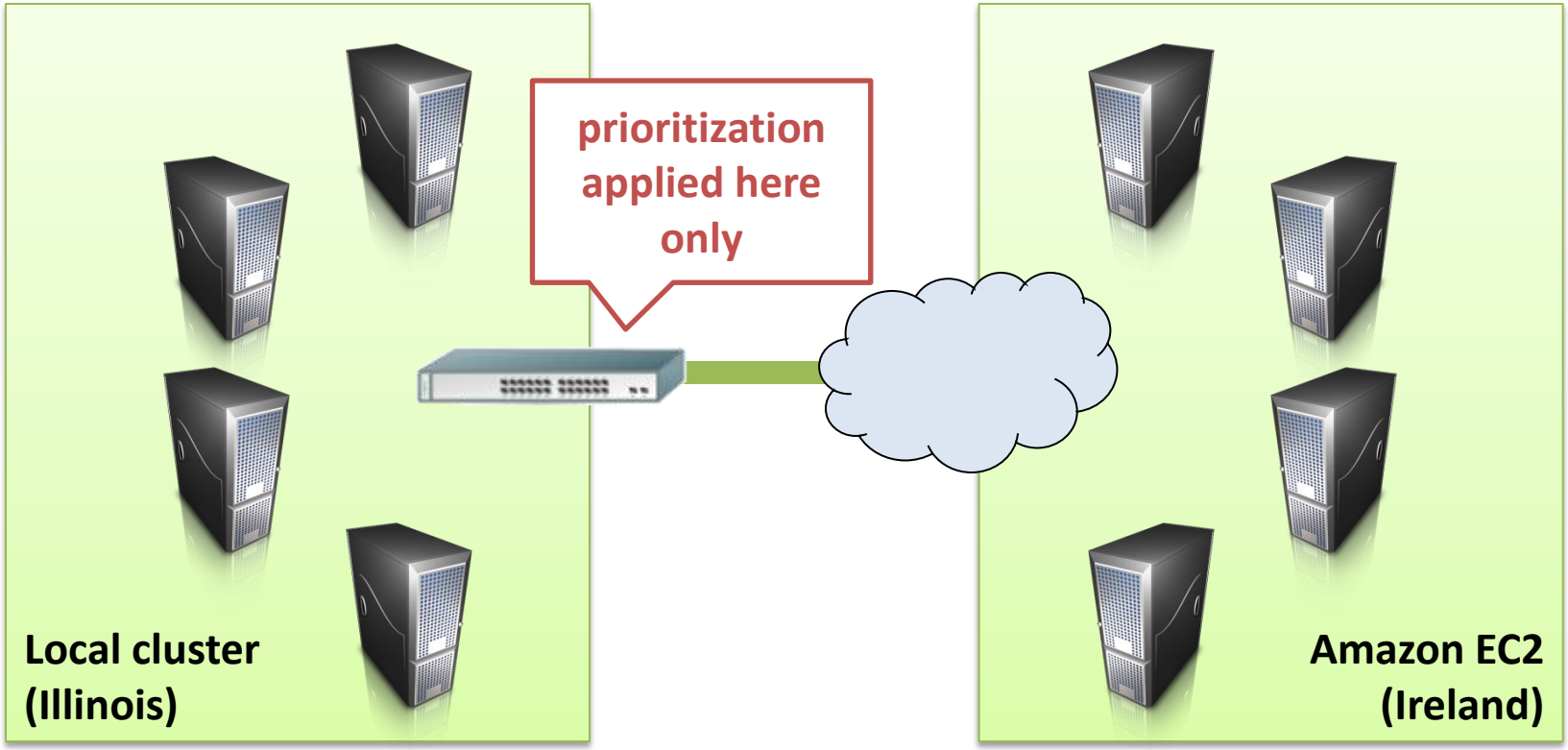


Evaluation topics

- Practical prioritization setup
- Delay with congestion
  - KV-store operations
  - Twitter clone web app operations
- Delay without congestion
  - Overhead of Vivace algorithms compared to traditional algorithms

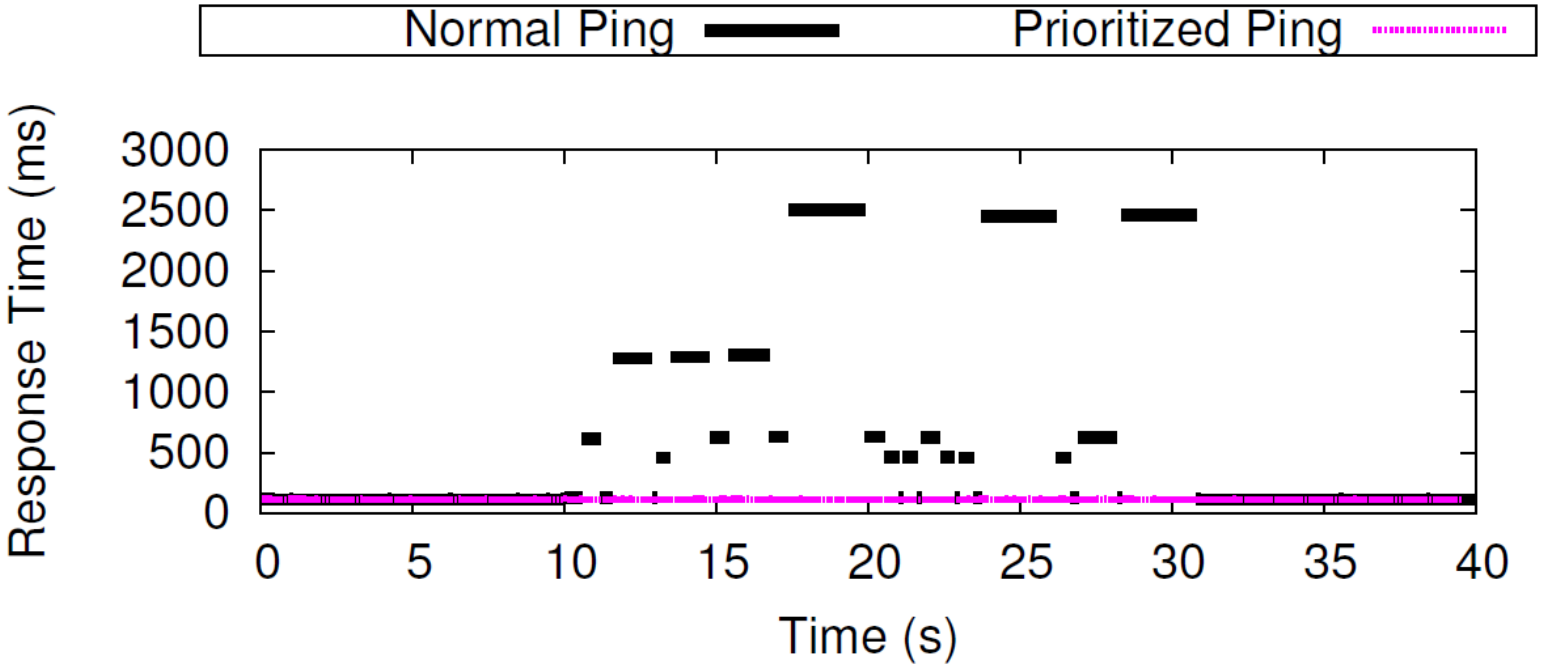
Evaluation setup

- Local cluster <-> Amazon EC2 Ireland
- DSCP bit prioritization on local router's egress port
- Congestion generated with iperf



Evaluation
Does prioritization work in practice?

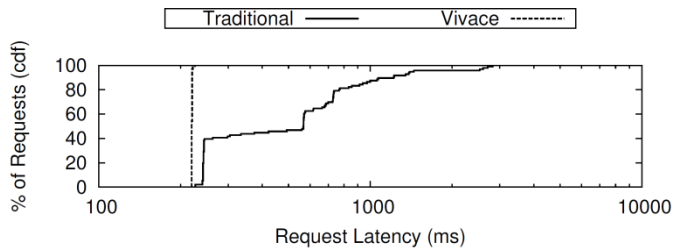
- Simple ping experiment
- Prioritized messages bypass congestion
- Local router-based prioritization is effective



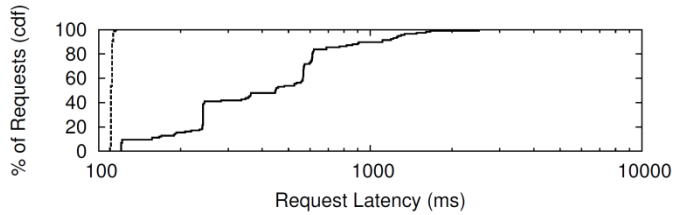
Evaluation

How well does Vivace perform under congestion?

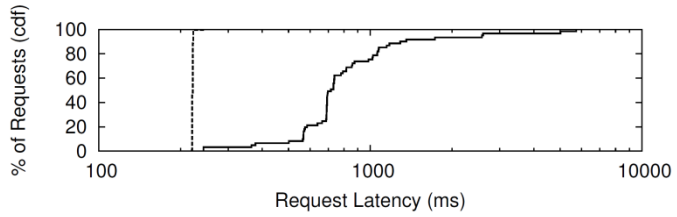
KV-store operations



(a) Read algorithms

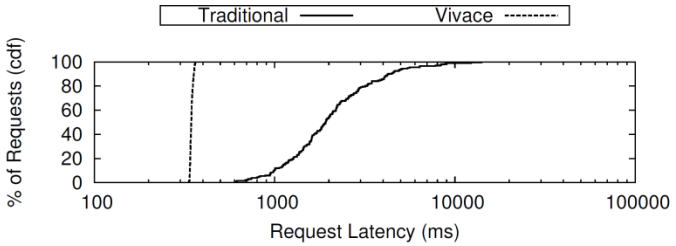


(b) Write algorithms

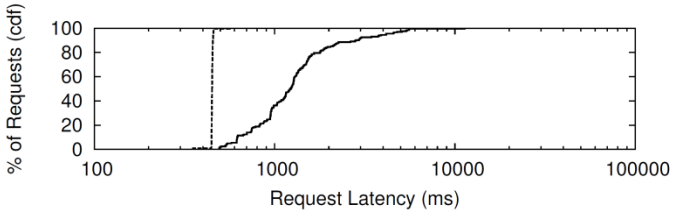


(c) State machine algorithms

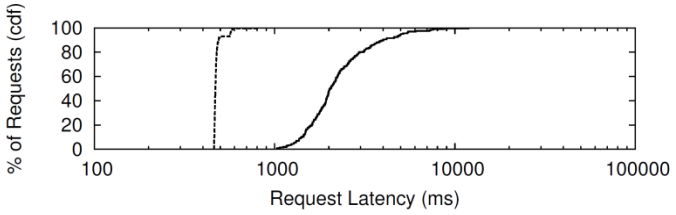
Twitter-clone operations



(a) Post tweet



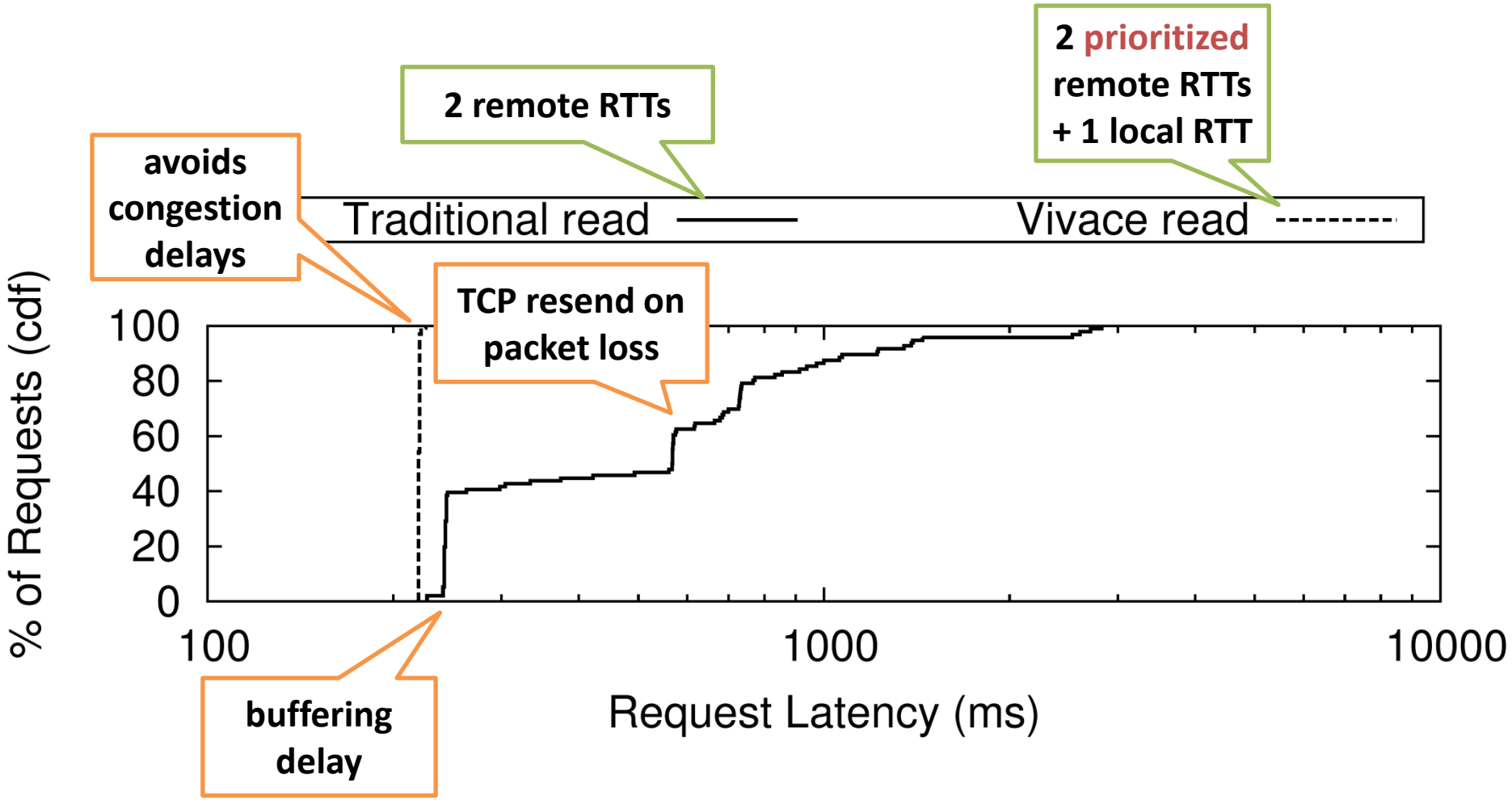
(b) Read user timeline



(c) Read friends timeline

Evaluation

How well does Vivace perform under congestion?



(a) Read algorithms

Evaluation

What is the overhead of Vivace without congestion?

- (Results in paper)
- No measurable overhead compared to traditional algorithms
- Extra message phases are not harmful

Conclusion

- Proposed two new algorithms
  - Read/write (simple, in talk)
  - State machine (more complex, in paper)
- Both algorithms avoid delay due to congestion by prioritizing a small amount of critical information, while
  - Still providing strong consistency
  - Keeping prioritized messages small
  - Avoiding delay overhead in absence of congestion
  - Using a practical prioritization infrastructure

- Careful use of prioritized messages can be an effective strategy in geo-distributed data centers